ACORN 1982
CORN
ORN
RN
1982 NIEUWS
NIEUW
NIEU
NIE
NI
N
jaar-gang NEGENTIEN
1982 TWEE
EN 1982
TACHTIG






++

Uitg.Fed.Acorn Computerclubs Ned/B.

1982

I N L E I D I N G

Dit boekje is een bundeling van de eerste jaargang van Acorn Nieuws, oestijds nog geheel verzorgd door de oprichters van de Acorn Computerclub: de heer G.H. Borghaerts en mevrouw M. Gruyters.
Met deze uitgave wordt beoogd deze eerste jaargang, waarin veel wetenswaardigs is opgenomen, voor later toegetreden leden opnieuw beschikbaar te maken.

Deze uitgave is tot stand gekomen door een lofwaardig initiatief vanuit de Acorn Computerclub Limburg.
Met name is dank verschuldigd aan mevrouw M. Ernst, die de gehele redactie, lay-out en het type-werk verzorgde.

Bij dit boekje behoort een cassette met programma's, welke verkrijgbaar is via de regionale bandjesarchieven.
Hiernaar wordt verwezen door, indien van toepassing, de naam van het betreffende programma bij de hoofdstuk-titel te vermelden.

Arno Millenaar,
voorzitter Federatie Acorn
Computerclubs Nederland/Belgie.

I N H O U D S O P G A V E

Toelichting:
++++++ : Nummer van Acorn Nieuws waaruit afkomstig.
"TELETYPE" : Naam van de file op het verzamelbandje van listings uit Acorn Nieuws 1982.

1 - 2 - 4 MHz.

Dit artikel gaat over hoe men met eenvoudige middelen de Acorn Soft- en Hardware matig kan omschakelen tussen een kloksnelheid van 1MHz en 2MHz. In de eerste experimentele fase monteer je een wisselschakelaar, zodat je al even kunt merken wat het verschil is in verwerkings snelheid. Hierdoor word je vanzelf warm voor beter werk wat hierna volgt, maar waar wel een paar uurtjes knutselen voor nodig is.

Het speelt zich voornamelijk af rond IC44 de 74LS393. Dit IC deelt het 4 MHz kloksignaal zodanig dat in de originele uitvoering de 6502 CPU een 1 MHz signaal krijgt.

Wil je het met een wisselschakelaar doen (fig.1) dan is dit wel leuk om even te proberen maar blijvend zal het geen succes zijn en wel om twee redenen: a)Na elke omschakeling is er een BREAK nodig. Dit komt door het denderen en het niet synchroon met de klokpuls schakelen, toevals treffers uitgezonderd natuurlijk.

b)Het is niet mogelijk om vanuit het programma de snelheid te wijzigen (soft schakelen).

De volgende eenvoudige schakeling (fig.2) voldoet al beter en heeft de twee voorgaande nadelen niet. De schakeling heeft slechts 1 IC nodig en wel de 7400. Toch is ook hier een nadeel en wel dit: na het aanzetten of na een BREAK weet je niet of je in de 1 MHz- of 2 MHz mode bent. Dit is te kontrolleren d.m.v. "CTRL G" (vergeet niet hierna op ESC te drukken). Indien de toon 2 maal zo hoog is als normaal dan is het uiteraard de 2 MHz mode. Deze schakeling maakt gebruik van een uitgangspoort van IC25. Door de selectlijn aan de PC-3 poort van de 8255 PIA te hangen is het mogelijk om vanaf het toetsenbord of vanuit een programma d.m.v. de volgende instructies de snelheid te veranderen zonder deze door een BREAK te moeten resetten. ?≠B002=≠EF voor de 2 MHz mode
?≠B002=≠E7 voor de 1 MHz mode

Het gebruik van deze poort PC-3 behoeft nog wel nader kommentaar maar daarover later.

Een leuk demonstratie programma geeft heel duidelijk de verandering van snelheid aan: DO P.$7;?≠B002=≠EF;P.$7;?≠B002=≠E7;U.0

Het resultaat zal een wisselende toonhoogte zijn.

Het navolgende schema (fig.3) kent geen problemen. Bij elke BREAK komt hij terug is de 1 MHz mode. Indien hij terug komt in de 2 MHz mode dan moet je de draadjes bij pin 11 en 13 van IC44 verwisselen. Hiervoor zijn 2 IC's nodig, kosten ca. f 3,50 en wel de 7486 (EXOR) en een 7473 (een JK flip-flop). Doordat er pas geschakeld wordt op de flank van de klok kan er geen fase fout worden gemaakt, ook ontstaan er geen reset problemen meer. Om deze schakeling te gebruiken heb je weer een poort nodig van 8255 PIA zoals dat ook bij de vorige schakeling gebeurde.

Om hierop terug te komen, ik ben mij bewust van het feit dat ik hier met mijn vingers in de standarisatie van de computer zit maar ik zag hier geen andere mogelijkheid of men moet een 2e 8255 monteren. Maar dit zou voor veel mensen een te grote stap zijn en hen ervan weerhouden omdit alles eens te proberen.

Als poort PC-3 (pin 17 van IC25) gebruikt wordt voor andere doeleinden dan die welke Acorn-computers hem heeft toebedeeld (het omschakelen van achterkleuren en contrast) dan moet LK 5 op de print (componentenzijde) worden doorgekrast. Om een onrustig beeld te voorkomen moet je pin 39 van IC31 (6847) met een 'pull-up' weerstand van ± 6Kb aan de +5V hangen. Wanneer je dit doet wordt het contrast van de computer ook sterker. Als dat niet gewenst is moet je pin 39 d-m-v- LK 6 aan de massa leggen.

fig.3

2MHZ PIN 11 IC44

4MHZ PIN 13 IC44

PIN 37 6502

J CL 1 K Q

J CL 1 K Q

+5V

IC25 PIN 17 8255

1 = 7473

2 = 7486



PIN 10 IC44

IC: 7400

PIN 11 IC44

PIN 37 6502

PIN 17 IC25 8255 select

PIN 37/6502

fig.1

2MHZ 1MHZ

PIN 11 IC44 PIN 10 IC44

Het volgende probleem is mogelijk: de snelheid van de 2114 geheugen IC's van het video-geheugen is niet goed. In de mode 2 t/m wil het bij plotten vaak een rommeltje geven. Hiervoor zijn 2 oplossingen:

a) De IC's vervangen door snellere types en eventueel meten of de IC's wel de broodnodige 5 volt krijgen

b) Voor elke PLOT,DRAW.... instructie de snelheid terug brengen naar 1 MHz en erna direct weer omhoog naar 2 MHz. Een voorbeeld hiervan:

regelnummer ?#B002=#E7;DRAWX,Y;?#B002=#EF of elke andere instructie

Bij dit alles is er wel een maar!

De temperatuur loopt op en de stroom consumtie neemt toe. Dat zijn maar kleinigheden zolang je erop let dat de IC's niet boven de 100° komen. Indien je geen geforceerde koeling gebruikt moet je de computer gewoon niet onnodig lang in de 2 MHz mode laten staan. Wat die koeling betreft, hier een suggestie: Bij de KWANTUM HALLEN wordt een axiaale kooi ventilator verkocht welke werkt op 12 V, maar ook wel op 5 V (eventueel de ongestabiliseerde spanning gebruiken). Deze ventilator kost maar f 6,50 en is op de zijkant van de kast van de Atom te monteren. Hiervoor moet je dan wel een langwerpig gat maken van ca. 10cm bij 2cm. en aan de andere kant een rooster of een paar gaatjes.

Arno Millenaar.

De T.V. als VIDEO-MONITOR

***

Een onderwerp dat brede belangstelling geniet, is het gebruik van de TV als MONITOR voor de computer. Dit betekent, dat men vanaf pen 9 en 10 van PL5, van de Acorn, met het computersignaal 'eruit gaat' (de zgn. Video-uitgang) en vlak voor de beeldversterker/synchronisator de TV 'ingaat'. Alleen de erg dure TV's hebben hiervoor vanaf de fabriek aangebrachte aansluitingen. Het schijnt ook te kunnen via de aansluiting 'Video-Recorder' van sommige TV's. Uw Acorn kan probleemloos op de TX Philips aangesloten worden (wanneer deze TV een 12 volt voeding-aansluiting heeft). De benodigde 'monitor-ingang' kunt u dan relatief eenvoudig zelf aanbrengen. Een dergelijke aansluiting heeft grote voordelen door een veel helderder, fijner gedefinieerd en stabieler beeld. Hoe u genoemde aansluitingen kunt aanbrengen, staat beschreven in Radio-Bulletin van 7/1981. Kunt u daar niet aankomen, dan informeren bij het drukwerkarchief. Weet u niet 100% zeker of uw TX geschikt is, informeer dan bij uw regio-hardware-man.

VIDEO CONVERSION

***

Using a portable television with the Atom video output.
When the UHF output is used with most portable tv's the definition will be very poor due to the narrow band-width of the RF and IF stages much less than the tv is capable of. This modification was made to my PERDIO portable TV with a great improvement to the definition but without affecting the picture quality when watching CROSSROADS.
The circuit has the advantage that the print on the PCB is not cut or altered, the extra components are just soldered to the underside of the PCB.A circuit diagram of the tv is a must.
With the Atom disconnected from the TV the AF117 in the extra circuit is reversed biased, so the screened lead has no effect on the performance of the circuit due to the extra capacitance when used as a television.
With the Atom connected to the TV the AF117 is operated as an emmiter follower circuit. Since the output voltage from the Atom is 1.5V and the voltage on the cathode of the video demodulator is 3V the diode will be reversed biased, so cutting out the noise from the TV RF stages. It is fortunate that the video output voltage is similar to the output from the demodulator and the polarity of the video signal from the Atom is correct. The AF117 transistor is not critical, any PNP RF type would be OK. Keep all unscreened leads as short as possible.
IMPORTANT NOTE: This modification should not be attempted on a non-isolated TV chassis. The power supply must use a mains transformer with secondary windings earthed. Schema op de volgende blz.

De voeding.

**

Bij de voeding moeten er naar meer punten gekeken worden. Verschillende klachten worden door de voeding veroorzaakt. Een klacht is dat het VDU beeld na enige min. na het aanzetten van de computer in ongunstige, danwel pas in gunstige zin veranderd. De fabrikant antwoorde dat de klacht is terug te voeren tot een veranderde voedingsspanning, en adviseerde om de verbindingsleiding in te korten. Op zich is dat juist. Het snoer is wat dun en bij een stroomsterkte van 1½ Ampere bij een 'full house' aan RAMs geeft een weerstand (in de verbindingsleiding) van bv. slechts ½Ohm reeds een spanningsval aan de computer van 0,7V. In eerste instantie vertoont dan de Modulator, dus het VDU kuren. Bovendien is de voedingsstekker niet al te best, dus mogelijk spanningsverlies. Men kan het

snoer (verwisseld voor een beter) ook aan de + en - vastsolderen.

Schema:
Video
Conversion.

Schema van een voeding
met 2 gescheiden
stabilisatoren.
Door:Peters uit Wehl.

A en B naar de afzonderlijke voedings-
kringen
v/d Atom

In: Van voedingsapp. ± 8V ongest.

Het dender probleem. **

Het denderen is meestal een hardware probleem. De toetscontacten moeten goed gesoldeerd zijn en de voeding moet goed zijn.

```
Software matig: !≠28F0=≠832003A2    C3 (inhoud ≠28F1) zorgt voor de
                !≠28F4=≠FE944CFB    vertragingstijd, zelf in te stel-
                !≠020A=≠F96E28F0    len.≠28F1=0, dan vertraagt het
                                    program 4 sec.
```

UITBREIDING van GEHEUGEN

***

Een steeds wederkerende vraag betreft de mogelijkheden van geheugen uitbreiding van onze Acorn-Atom. Een simpel antwoord hierop zou zijn, dat je met uitbreiding nét zover kunt gaan als je portemonnaie reikt en je voedingsapparaat trekken kan. Elke 6502 in elke machine heeft met z'n 6-bit 64K van zichzelf. Hoe gebruikersvriendelijk de machine is, hoe meer daarvan aan ROM wordt opgegeten. De rest is RAM; bij ons 12K. Relatief weinig RAM pleit dus geenszins tégen een computer! Aan vele computers, zeker aan de onze, kun je extra blokken RAM of ROM hangen, wil je in totaal méér dan 64K, dan wordt het 'dubbel parkeren'. Iemand heeft er bv. al 128K extra RAM aanhangen. Je kunt aan de Acorn ook weer een nieuwe 6502 aanhangen en tot het oneindige multiplexen. Momenteel kunnen uitbreidingen gaan in 'statische' RAM, zoals de 2114's die er al inzitten; in 'dynamische' RAM's die heel wat goedkoperzijn maar extra 12V voeding vragen; in EPROMs voor het opslaan van steeds voorhanden routines en in zgn. pseude-ROMs, die je kunt gebruiken als gewone RAM,doch waarin de inhoud kan worden behouden door er met knoopcellen min. 2 volt op te houden na uitschakelen van de computer. Geschikte typen hiervoor zijn voor ons de 6116 en 8416.

Een vaak betere vraag dan 'hoeveel kán eraan' is: 'Hoeveel heeft u nodig?' Bij alle vragen om uitbreiding dienen we niet te vergeten, dat de grote meerderheid van Atom-hobbyisten aan de voorhanden 2 blokken van 5½K al jaren genoeg hebben. Temeer daar wij met onze ?16 op een uniek manier in de machine kunnen rondspoken. Maar toch, een paar K extra RAM is wel zo plezierig.

Maar hoe?

a) u koopt, of maakt, een geheugenkaart, bv. de clubversie.

b) u begint vast met min. kosten de reeds aanwezige geheugenruimte met een paar K extra RAM bij te plakken (het lage geheugen met 1K tot de volle 6K, het hoge geheugen met 2K tot de volle 8K).

Oplossing b) kunnen we nader uitwerken:

'laag' geheugen

2800 2C00 3000 3400 3800 3C00 4000 8000

16K

'hoog' geheugen

8000 8400 8800 8C00 9000 9400 9800 A000

Gearceerd //// in volle Atom aanwezig geheugen in RAMs
Zwart eenvoudig bij te plakken RAMs
Gespikkeld aan te vullen met bv. 6116's

Er ontstaat dan een aaneengesloten geheugenblok van 30K waarvan 16K naar wens te gebruiken als RAM,ROM of EPROM. Bepaald niet onaantrekkelijk.

Tot dusverre spraken we over 'bijplakken' van RAM's. Als u het op de simpelste, goedkoopste en toch degelijke manier wilt doen, gaat het letterlijk om plakken. De toe te voegen 6 RAM's 2114 worden dan inderdaad gewoon op de rug van 6 reeds aanwezige RAM's 2114 geplakt, n.l. op de IC's 32,33,34,35 (hoog) en IC 10,11 (laag).



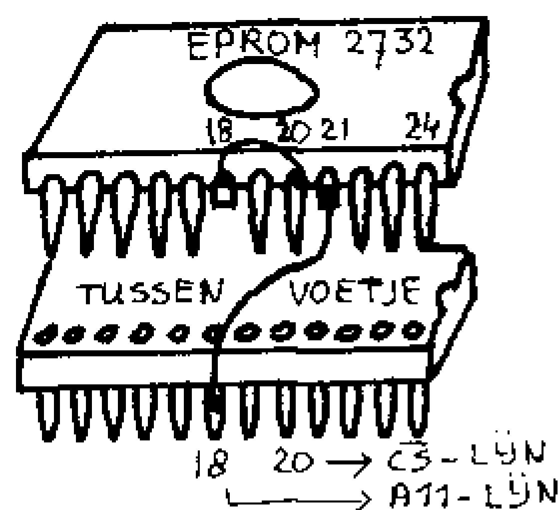
In Engeland noemt men dit 'Pig Packing', varkens stapelen. De pootjes van het bovenste varken zitten normalerwijze reeds klemmend om de overeenkomstige pootjes van de onderbuurman en worden hieraan vastgesoldeerd. Of,

als u dit griezelig vindt, vastgeplakt met bv. BISON ELECTROKIT.
Dit doet u met alle pootjes op één na, n.l. pootje nr.8, d.i. de Chip Select. De pootjes nr.8 van de bovenvarkens soldeert u NIET aan de onderbuurman maar buigt u iets naar buiten.
U steekt nu IC 10 en 11, ieder beladen met een 'pig' weer terug in de sockets. Die RAM's doen het dan weer alsof er niets gebeurd is. Nu verbindt u met een draadje beide pootjes 8 van de bovenvarkens en verbindt dit verbindingsdraadje met pin 7 van IC 6. Daarmee is het lage geheugen klaar.
Met IC 32 en 33 doet u hetzelfde, bij elk een 2114 erbovenop, alle pootjes doorsolderen of lijmen met de onderste, op de pootjes 8 na. Na terugsteken van de IC's 32 en 33 verbindt u beide losse pootjes 8 van de twee bovenvarkens en verbindt dit verbindingsdraadje met pootje 7 van IC 30.
Tenslotte haalt u IC 34 en 35 eruit, voorziet ze op precies dezelfde manier van een bovenvarken. Na terugsteken van de IC's 34 en 35 verbindt u weer de beide pootjes 8 van de bovenvarkens en verbindt dit verbindingsdraadje met pootje 9 van IC 30.
Hiermee is de operatie klaar. U heeft nu twee geheugenblokken, n.l. een laag geheugenblok van de volle 6K (de tekstpointer blijft gewoon op ≠29) en een hoog geheugenblok van een volle 6K. Alle zaken voor het selecteren van de bijgeplakte RAM's en de lees/schrijf leidingen zaten al op het board. U heeft ze nu gebruikt. Om niet dóór het board te hoeven boren kan men de verbindingsdraadjes naar IC 30 pin 7 resp. 9 rechtstreeks aan de boven uitstekende delen van de poten van dit IC solderen. Doe dit degelijk! CS lijnen mogen nooit los zitten!

****

UITBREIDEN met EPROM 2732.

Met de printsporen op het moederboard van de Acorn-Atom is de EPROM 2532 zonder meer compatibel (uitwisselbaar). De pin-configuratie van de 2732 is anders en dient te worden aangepast aan het Atom-board.
Toelichting: De ADRESlijn A11 op het board gaat naar de voetpin 18 van alle 24-pins ICs. Bij de 2732 moet deze echter naar pin 21. De Chip-Select lijn op het board is pin 20 ($\overline{CS}$) De 2732 heeft echter 2 aan/uit schakelaars, n.l. 18=$\overline{CE}$ (Chip-Enable) en 20=$\overline{OE}$ (Output Enable). Beide zijn 'aan' als ze LAAG zijn en beide zijn 'uit' als ze HOOG zijn. Kunnen dus dóórverbonden.
LET OP: Van alle 24-pin IC's op ons board zijn ALLE pinnen (overeenkomstige) vast doorverbonden op één na, n.l. de Chip-Select lijn. Deze lijn (20) komt per IC apart uit een decoder en is de enigste lijn, die uitmaakt of een IC functioneert (als de lijn laag is) of uitgeschakeld (lijn hoog).



Eenvoudigste 'ombouw' van EPROM 2732 naar socket van IC24 via een tussenvoetje. Socket 20 gaat dus naar EPROM 20 én 18. Socket 18 gaat naar EPROM 21. Pootjes 18 en 21 van de EPROM kort afknippen (of naar buiten buigen.)

We kunnen dus, als we willen, IC's in IC-voet 24 of bovenop de Floating Point, als varkens stapelen, op voorwaarde, dat we de pinnen 20 uitbuigen.
Dit varkens-stapelen (Pig Packing) deden we al met de RAM's 2114. Bij een 24-pin IC moet nu de Chip-Select aan pin 20/18. Het verschil is, dat we bij het RAM-stapelen op het board DECODER-lijnen over hadden voor de CS Bij de 24-pin IC's echter niet. Om nu toch meerdere bruikbare EPROM's om beurten te kunnen gebruiken, kunnen we de Chip-Select-lijn naar de

verschillende 24-pin IC's omschakelbaar maken. Met een 2-3 of 4 keer om-schakelaar of met zgn. dip-schwitches.
Let echter terdege op 2 voorwaarden: 1)als u dip-schwiches gebruikt mag u nooit twéé 24-pin IC's tegelijk aan de Chip-Select-lijn schakelen en 2)de chip-select ingang (bij de EPROM 2732 dus pin 18 aan 20) mogen bij niet gebruiken nooit loshangen maar moeten gedefinieerd naar Hoog. U doet dit het gemakkelijkst door alle 24-pin IC's te voorzien van zgn. PULL-UP weerstanden, d.i. een 5K6 weerstand van genoemde ingang naar pin 24 = 5Volt. In gebruik trekt de Chip-Select lijn deze ingang wel, ondanks de weerstand, naar Laag. In niet-aangesloten toestand vliegt de ingang echter door die weerstand naar Hoog. U kunt daar kleine 1/8 Watt weerstandjes voor nemen.
U kunt natuurlijk ook afzien van solderen aan de IC's voor tussenvoet en weerstanden, door een net printje te maken en daar alle zaken via de bedrading in te verwerken. Dat printje kan dan in de kast, met de om-schakeling 'naar buiten' gevoerd. Of de hele IC24 socket met een flat-kabel 'uitvoeren' naar een Low Extraction Force-socket, waar u de beno-digde Utility-EPROM's naar believen kunt omwisselen.

EPROM 2732 in IC24 voet. *
Van ZERO in Bergschenhoek kwamen de gegevens voor een 2732 in socket IC24. Pootnummering van bovenaf gezien op chip of voetje. Beginnen bij het 'slotje' en linksdraaiend. Via 1 of gemakkelijker 2 tussenvoetjes op elkaar, moet:
pin 20 van de socket naar pin 20 en 18 van de EPROM
pin 18 van de socket naar pin 21 van de EPROM
pin 21 van de socket blijft vrij, afisoleren.
Het adres van IC24 is A000-AFFF = 4K.

Over LADEN en LOSSEN van CASSETTES en BANDJES

**

De informatie op het bandje, bestaat uit de pulsen van afwisselend 1200 en 2400 Herz. Deze wordt in uiterst fijne streepjes dwars op het bandje geschreven. Bij het aflezen van deze fijne magnetische streepjes heeft de spleet in de schrijf/afleeskop van de 'eigen' recorder dezelfde stand (dwars op de band) als bij het schrijven. Het is namelijk dezelfde kop. En als deze bij het schrijven niet helemaal haaks op de band staat, maar bv. een haartje scheef, staat hij bij het aflezen precies even scheef. Geen probleem.
De stand van de kop dwars op de band, ideaal dus 90°, heet de AZIMUTH. Wat er gebeurt, wanneer een bandje dat geschreven werd op een recorder met de kop een haartje naar links, wordt afgelezen op een recorder met de kop een haartje naar rechts kunt u zich wel voorstellen. In een situatie als deze worden pulsen die bedoeld zijn als BLOKVORM, gelezen als iets tussen SINUSVORM en ZAAGTAND.
Genoeg theorie: Wat doen we aan een scheve kop? Bij elke recorder is de azimuth instelbaar met een stelschroef links of rechts van de kop, nl. dié schroef, waar het veertje onder zit. Meestal zijn de schroeven gelakt. De beste en simpelste manier is om bij de winkelier of bij een kennis met een recorder van hoge kwaliteit een bandje te laten maken, waarop een constante fluittoon staat van een toonhoogte tussen 2500 en 3000 Herz. Thuisgekomen sloopt u de deksel van de recorder af zo dat u met een klein schroevendraaiertje bij de azimuth-stelschroef kunt komen. Het is de bedoeling dat u het stelschroefje verdraait terwijl het fluitbandje speelt. Op het moment, dat de geproduceerde fluittoon op z'n hardst klinkt en uw vrouw vraagt of u niet een andere hobby weet, staat de kop goed. Wanneer uw recorder wat zweeft is dat niet zo erg. De computer slikt dat wel.

Het WASSEN van bandjes. **

Heeft u kans gezien om een programma met pijn en moeite te LOADen en wilt u het daarna 'GEWASSEN' vanaf de computer op een nieuw bandje zetten, dan blijkt soms, dat dit niet gaat door verminkingen en/of blokkeringen, die in de voorgeschiedenis zijn ontstaan of aangebracht. U kunt het programma dan wel RUNnen, maar niet copiëren; niet door een audiocopie omdat het bandje bv. te slecht is en niet via de computer vanwege verminking of blokkering.

Met *CAT vindt u	"program"	Startadres	Executieadres	Bloknr.	Vulling
bv. als 1e regel	PROGRAM	2900	C2B2	0000	FF
als laatste regel	PROGRAM	3300	C2B2	000A	A6

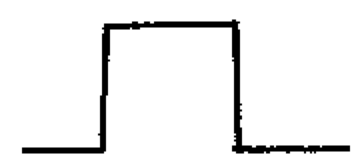
Voor *SAVE heeft u nodig: "naam" Startadres Eindadres Executieadres
Het startadres weet u, in dit voorbeeld dus 2900
Het executieadres ook, in dit voorbeeld dus C2B2
Het eindadres vindt u door: van het laatste blok bij *CAT het beginadres te nemen (hier dus 3300) hierbij de vulling van dat blok (hier A6) plus 1 byte (altijd bij *SAVE) bij op tellen (hier dus in totaal 33A7).
In dit voorbeeld doet u dus *SAVE "PROGRAM" 2900 33A7 C2B2. U heeft dan weer een 100% fris bandje van uw geliefde program.

*FLOAD **

Bij *FLOAD doet u het volgende (dit staat niet duidelijk in het boek): direct na SUM ERROR stopt u de band, dus vóór de 'leader' volgend blok. Dan typt u in *FLOAD"NAAM" (ret) en vervolgt het laden met volgend blok. Hetzelfde met evt. volgende SUM ERRORs. Na laden LIST, RUN zal het wel niet doen.

Zijn er ook te goede recorders? ****

In Acorn Nieuws nr.2 hadden we het over Azimuth-problemen, die zich, vooral bij eenvoudiger recorders, kunnen voordoen. Aan de andere kant is vrij algemeen bekend, dat een 'dure' recorder voor computerdoeleinden ook niet automatisch de beste resultaten geeft. Veelal zijn die 'te goed'. De oorzaak hiervan zit in de golfvorm, die de computer bij 'save' aflevert. Globaal omschreven schakelt de computer aan z'n uitgang alleen van Hoog naar Laag en omgekeerd en dat razendsnel en zou dus blokgolven leveren. Een zuivere blokgolf is in theorie een golfvorm, die samengesteld is uit de grondfrequentie en alle boventonen in gelijke verhouding.
Het zijn juist die boventonen, waar de computer bij teruglezen van de band niets van hebben moet. In de praktijk is bovendien de oorspronkelijke golf geen zuivere blokgolf maar een blokgolf met 'rafels' die duiden op extra aangedikte boventonen. Een simpele recorder verwaarloost die boventonen, slijpt ze eraf. Een dure recorder is duur omdat deze wel zuinig is op boventonen. We willen toch Hi-Fi. De Hi-Fi recorder levert de oorspronkelijke blokgolf terug als een pyramidevorm met trapvormige flanken. De computer kan bij lezen hiervan de trappen 'triggeren' als extra pulsen; telt te veel en geeft dan SUM ERROR.
Om van deze problemen af te komen kunnen we voor een paar dubbeltjes een eenvoudige 'low-pass' schakeling maken. Dergelijke filters, die dat doen zitten al in de uit- en de ingang van de computer. In de praktijk blijkt die filtering echter niet afdoende. Je kunt deze wel wijzigen maar je kunt ook de cassette-kabel doorknippen en daar een extra filtertje tussenzetten. Je kunt dan de computer eventueel 'inbedrijf' laten, continu xxxx laten produceren (Manual blz.8) en knutselend aan het filter op een oscilloscoop zien wat het resultaat is.
Die resultaten zal ik voor u beschrijven:
Van de cassette-kabel werd de leiding ván computer náár recorder ergens opengemaakt. Je kunt dan de binnendraad (signaalleiding) en de afscherming (aarde) wat uit elkaar trekken. Die binnendraad verbinden we met de afscherming d.m.v. een condensatortje. Als bekend 'spert' een condensator gelijkstroom doch 'geleidt' wisselstroom. Hij geleidt die wisselstroom destemeer, naarmate de frequentie van die wisseldtroom overeenkomt met de resonantie van de condensator. Om hoge tonen te laten afvloeien hebben we een klein condensatortje nodig met een resonantie frequentie die hoger ligt dan 2400 Herz (dat signaal moeten we juist sparen).
Nemen we als condensatortje ca. 20 nF, dan zien we op de osciloscoop, dat van de oorspronkelijke rafelige computer-blokgolf de linkerhoek wordt afgerond. Maken we het filter compleet volgens tekening, dan wordt de rechterhoek afgerond. Er resteert dan nog een hoekigheid aan de top, doch wanneer dit signaal op een goedkoop bandje is geschreven en wordt teruggelezen, blijkt die hoekigheid praktisch verdwenen. We naderen dan de zuivere sinus. Aangezien een filter nooit geheel selectief is op één frequentie, verdwijnt er in dit filter ook iets van de 1200 Herz toon en nog iets meer van de 2400 Herz. We kunnen het filter nu verfraaien door zgn. klippen met 2 tegengesteld gerichte silicium diodes bv. 2x 1N4148. U zult wel om de verliezen in het filter te compenseren de volumeregelaar bij 'record' verder moeten opendraaien. De resultaten van het beschreven filter bleken bij mij alle verwachtingen te overtreffen.
Wanneer men via dit filter de xxxx op de band zet, blijkt de computer bij DO P.$BGET A;U.0 (zie Manual blz.8) uitermate tolerant. Vanaf praktisch afknijpen van de volumeregelaar tot 2 à 3-voudig oversturen blijft de computer rustig xxxx schrijven.
Voor tekeningen van signalen en de schema's zie volgende bladzijde.
Dit artikel en de artikelen van vorige pagina zijn van G.Borghaerts.



zuivere blokgolf



computer blokgolf



wordt zó door dure recorder
op goedkope band geschreven



oorspronkelijk
computerblokgolf



ná eerste
condensator

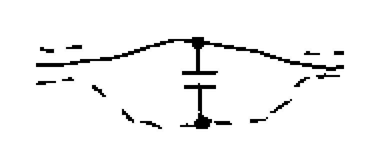


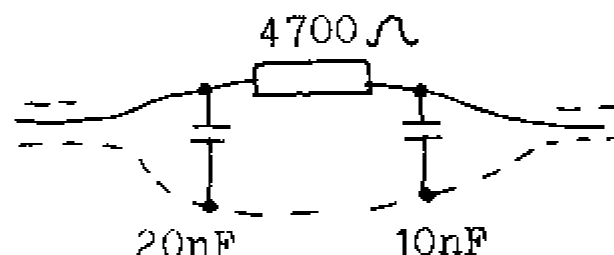
ná hele
filter

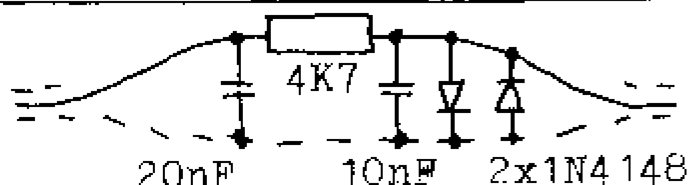


ná teruglezen
van 't bandje





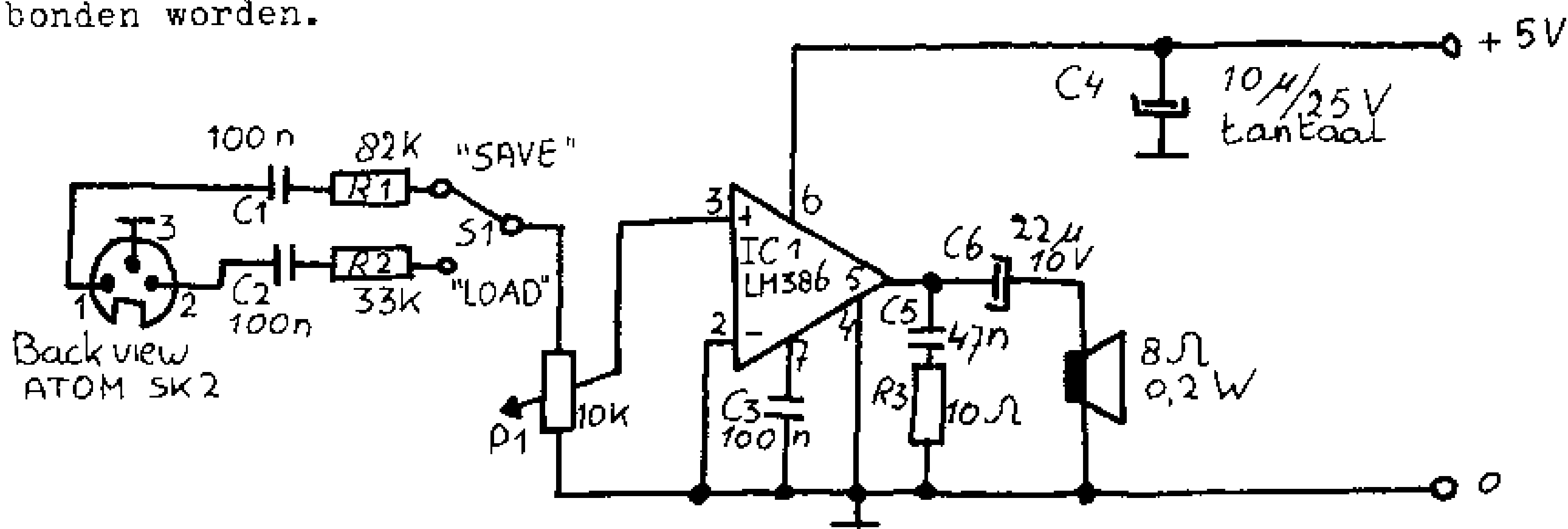




VISLOAD van Arie Marchal "VISLOAD" **
Dit programma, tevoren geladen en ge'run'd maakt zichtbaar laden mogelijk van volgende programma's.

ALLCOS. "ALLCOS" *********
SAVE en LOAD routines. Altijd zichtbaar op het scherm. Door A.Marchal.

Luistervink. ******
Meeluisteren bij LOAD en SAVE dat kan. We horen dan of de bitjes op de cassette of in de computer terecht komen. De ingangs-condensatoren C1 en C2 worden <u>in</u> de computer op de DIN-plug SK2 aangesloten. De waarden van R1 en R2 zijn richtwaarden. R1 en R2 moeten zodanig gedimendioneerd zijn dat bij omschakelen van S1 (LOAD/SAVE) het signaal even hard klinkt. In rust moet S1 in stand LOAD staan.
a) Door de grote versterking van IC46 in de ATOM heeft men bij laden maar een klein signaal op de ingang van SK2 nodig. De meeste cassettedecks geven een vrij groot signaal af, wat tot oversturing kan leiden. Een eenvoudige weerstandsdeler lost dit op.
b) De uitgangssignalen van stereo-decks mogen NOOIT met elkaar doorverbonden worden.



RS 232- Interface voor de Acorn Atom

[redacted]

Inleiding:
Om data te verzenden van een computer naar een randapparaat of andere computer zijn er twee verschillende methoden mogelijk. Dit zijn parallelle overdracht en seriële overdracht. Bij parallelle overdracht worden voor een byte data 8 verschillende lijnen gebruikt als ingang of uitgang van de computer. Bij seriële overdracht is dit slecht 1 enkele lijn.
De Atom beschikt al over de mogelijkheid om data parallel te verzenden m.b.v. het VIA IC (6522). Om een veelheid aan bedrading te beperken kan men seriële overdracht toepassen. In zijn meest eenvoudige vorm, is dit slechts een 3-draads verbinding van computer met randapparatuur.
Om de data van de databus (parallel) om te zetten in seriële data kan men een enkele poort van de VIA gebruiken, echter er is dan een vrij groot programma nodig wat dan ook nogal gecompliceerd van opbouw is. Voorts krijgt man problemen bij het transporteren van data in 'full duplex' (zenden én ontvangen over een enkele verbinding op hetzelfde moment). Een betere methode is om deze functies te laten verzorgen door een enkel IC. In de 65xx serie is er een dergelijke chip aanwezig. Dat is de 6551 ACIA (Asynchronous Communications Interface Adapter). Dit IC noemt men ook wel een UART, zoals de 8251 van INTEL die dezelfde mogelijkheden heeft voor computers met 8080 of 8085 processor.
De 6551 bevat alle benodigde electronica om data van parallel naar serie om te zetten volgens een programmeerbare codering, en vice versa. Ook kan de IC serie data verzenden en ontvangen op 16 verschillende snelheden (baudrates).
Worden alle zend- en ontvangsignalen van deze ACIA nog omgezet in +12/-12V niveaus dan heeft men een RS 232-C interface. RS 232-C is een Amerikaanse normalisatie voor seiële transmissie van computers met allerlei randapparatuur, de Europese equivalent is de V24 norm. Het verschil tussen beide is de connector die men toepast, RS 232-C gebruikt een 25 pol. D-connector met genormaliseerde aansluiting en de V24 schrijft geen bep. type connector voor.
De Schakeling:
De hieronder beschreven schakeling bevat een 6551 met interne baudrategenerator, een bijbehorend kristal, voor alle i/o lijnen optionele inverters (via stropjes te selecteren), lijn-drivers en -receivers om ervoor te zorgen dat de signalen op RS 232-C niveau gebracht worden. Alle verbindingen van de ACIA met de PL8 bus van de Atom zijn voorzien van Pull-up weerstanden om verrassingen te voorkomen. Ongebruikte inverters zijn via weerstanden met de 5V voeding verbonden, de voedingsspanning van de verschillende IC's zijn ontkoppeld bij de IC's en de +12V/-12V voeding is nog eens extra afgevlakt teneinde zo zuiver mogelijke voedingsspanningen te verkrijgen.
Voorts vindt u in het schema enkele IC's die tot doel hebben om het niveau van de ingangen en uitgangen van de ACIA op het genormaliseerde spanningsniveau van RS 232 te brengen (+12V voor een '0' en -12V voor een '1').
De voeding maakt de schakeling compleet. Deze hoeft slechts de beide IC's 2 en 3 te voeden.
De complete hardware staat beschreven in een artikel dat in het drukwerkarchief is opgenomen.
De interface kan geprogrammeerd worden voor de volgende functies:
baudrate (50,75,110,135,150,300,600,1200,1800,2400,3600,4800,7200,9600, en 19200 baud)
aantal stopbits (1,1½ en 2 stopbits)

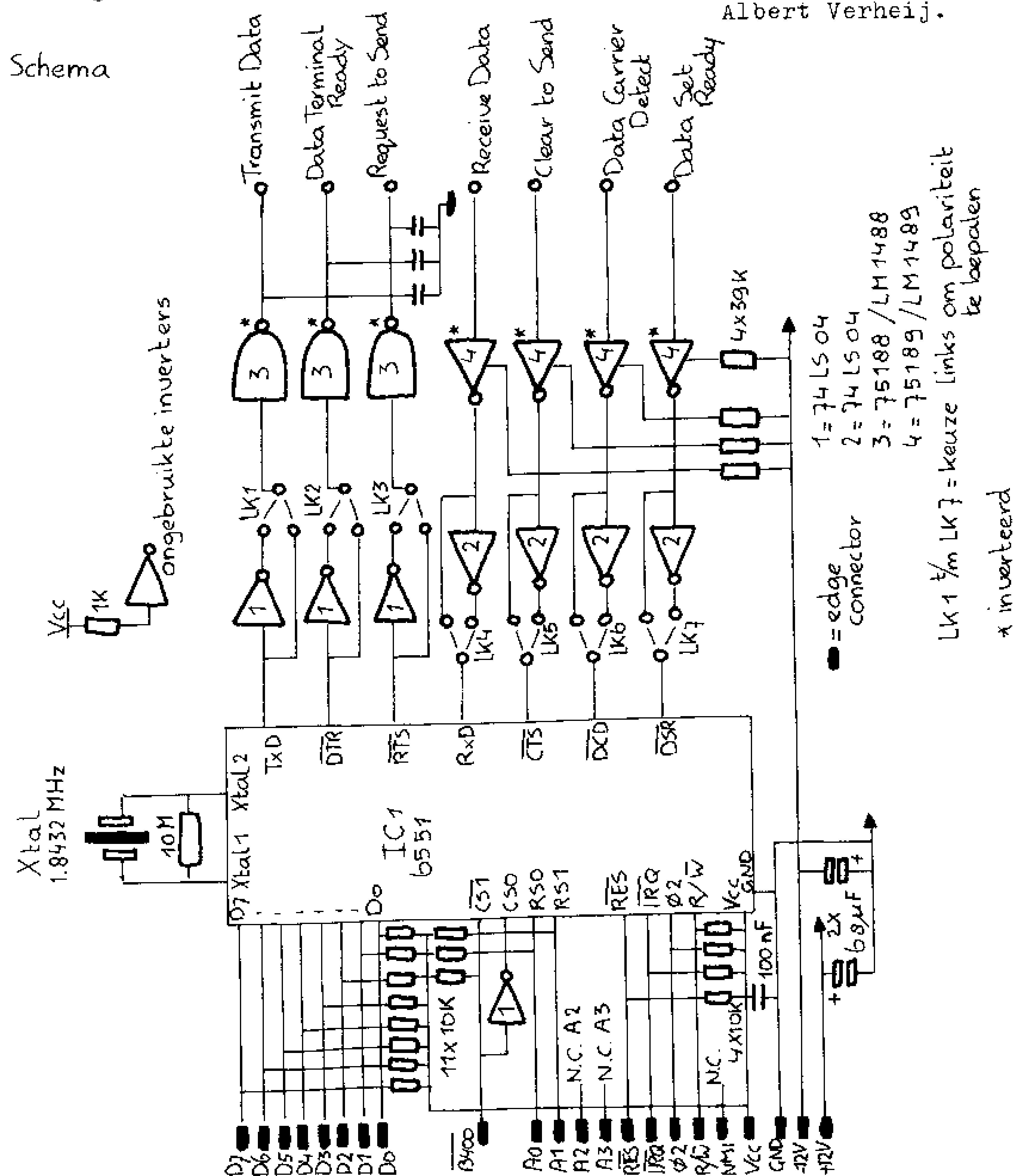
data woordlengte (5,6,7,8 bits overdracht)
echo-toestand, interrupt aan/uit, ontvanger aan/uit, zender aan/uit, reset.

Bij de in het drukwerkarchief aanwezige pakket documentatie is ook een programmavoorbeeld dat de ACIA kan sturen. Tevens is er een print lay-out bij.

Tot slot dan nog wat over de aansluiting op de Atom. Er wordt gebruik gemaakt van de nog vrije connector PL8. Deze connector bevat de adreslijnen voor het bereik ≠B400 - ≠D7FF. De interface bestrijkt slechts de adressen ≠B400 - ≠B403.

Ikzelf heb de schakeling nu ingebruik om m.b.v. een MODEM per telefoon te communiceren met een andere computer die 40 km verder staat. Ook heb ik al een andere computer DIRECT gekoppeld om op de respectabele snelheid van 9600 Baud gegevens met elkaar uit te wisselen. En dan hebben we het nog niet eens over alle mogelijke randapparatuur.

Albert Verheij.



Uitbreiding van schakelinterfacing

***

Een globale orientatie:
Voor schakeldoeleinden zijn input-output mogelijkheden nodig. Men noemt dit in het algemeen I/O poorten.
Hoe staat het daarmee bij de Acorn-Atom?
Binnen de organisatie van de Atom is alle I/O ondergebracht in het 4K adressenblok #B000 - #BFFF.
Het IC 49, een LS139, verdeelt dit blok in 4 blokken van 1K.
Het eerste blok B000-B3FF gaat naar IC 8255= IC 25. Deze schakelt (leest) het toetsenbord, schakelt de cassette I/O en nog wat zaken.
Het tweede blok B400-B7FF gaat naar de printconnector PL6. Aan deze connector komen tevens 4 adreslijnen (A0-A3), de ChipSelect CS#B400, de volle databus en controlbus (R/W,Clock,Inter.,Reset) en de 5V-0V. Dat is volledig voldoende om aan deze connector b.v. een VIA 6522 te hangen met z'n 2x8-bits I/O poorten,handshake,etc. Dat kan, maar hoeft niet. Er kan ook b.v. 4 keer een 8255 aan of een IC 6850 t.b.v. een seriele uitgang voor printers, een telefoonmodem, joy-sticks, A/D convertors etc.
Het derde blok B800-BBFF gaat naar IC 1. Dat is een 6522 VIA waar, aan één van de poorten, de centronic's printers aanhangen via het buffer IC 50. P16/7 geeft o.a. alle poort bits nog een keer. Nu echter ongebufferd.
Het vierde blok BC00-BFFF kan worden betrokken van pootje 7 van het genoemde IC 49 d.i. de Chip Selct BC00-BFFF. Voor hobbyisten, die hongeren naar poorten, zoals ikzelf b.v. om er modelspoor mee te sturen met z'n vele schakelingen, biedt deze adresband van 1K de gelegenheid om er nog eens een IC 6522 VIA mee aan te sturen, de rest van de lijnen hiervoor kan men eveneens van PL6 betrekken.
Zijn b.v. de 3 mogelijke IC's 6522 met hun totaal 60 poorten nog niet genoeg (b.v. bij verfijndere modelspoorschakelingen) dan is er nog de mogelijkheid van echt grof geschut. Uit de laatste adresband is het mogelijk tot 64! IC's 6522 aan te sturen. Hiertoe dient men dan uit de adreslijnen A4...A9 in totaal 64 CS lijnen te decoderen met b.v. 8 maal een 74LS138 en wat extra logica. 1300 I/O bits!
Dit globale overzicht kan u een idee geven!

G.Borghaerts.

Aansluiting van printers

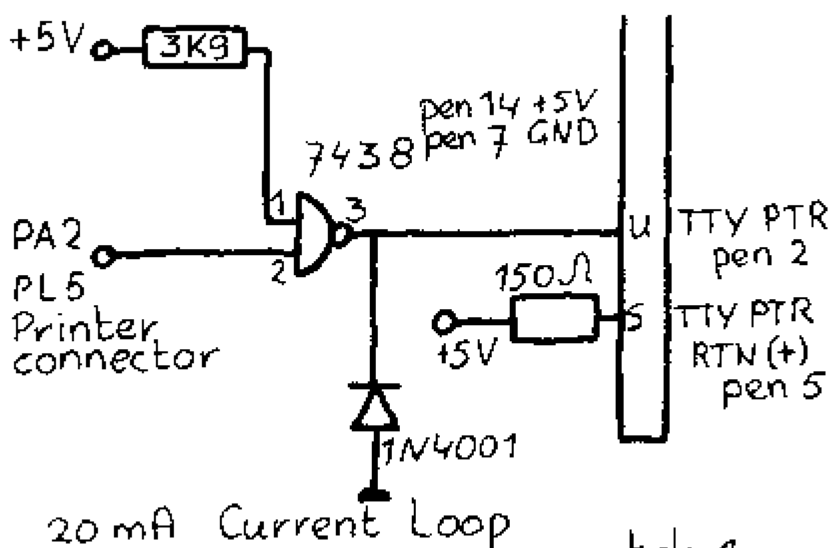
Aansluiting Seikosha GP 80M. *
De Seikosha GP 80M wordt aangesloten aan de ACORN aan de buffer LS244.
Zie ACORN schema linker bovenhoek: de IC's 1 en 50 moeten aanwezig zijn.
De volgende signalen moeten doorverbonden:

Strobe printer aan strobe PL5
Data 1 " " data 0 "
Data 2 " " data 1 "
Data 3 " " data 2 "
Data 4 printer aan data 3 PL5
Data 5 " " data 4 "
Data 6 " " data 5 "
Data 7 " " data 6 "
Ackn. van de printer aan ackn. PL5
Busy " " " " busy PL5.
Data 8 van de printer naar logic 0 van de printer = pin 12. Of voor grafische mogelijkheden zie volgend artikel.
Tussen de signaaldraden steeds een aardlijn!

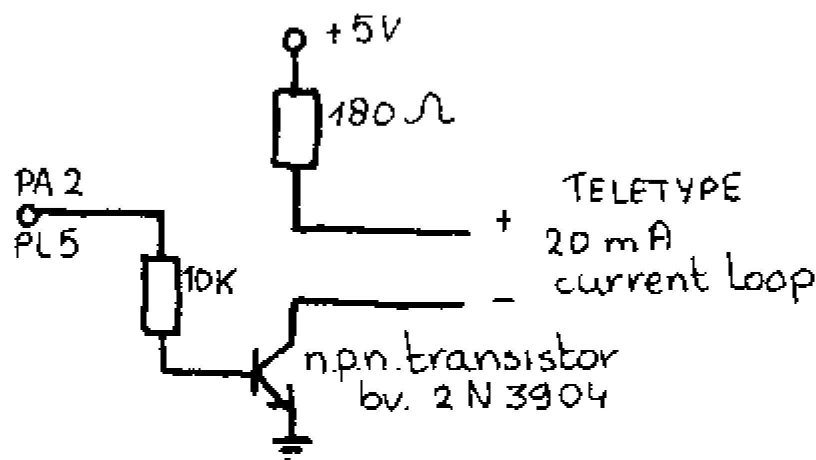
Aansluiting EPSON 82. **
De EPSON type 82 is evenals de Seikosha een printer met Centronics (parallel) aansluiting. De aansluiting van deze printers komt dan ook overeen. Op een punt na: data 8 van de printer moet niet naar logic 0 pin 12 van de printer maar naar logic 0 pin 16 van de EPSON printer.

Aansluiting TELETYPE. "TELETYPE" ***
Arie Marchal kwam in het boek 6502 Software design de aansluiting voor een teletype aan een AIM65 tegen. Hij werkte dit om voor de Acorn, zie hier (en op het bandje) het resultaat.
De printer kan aan en af gezet worden met ?AF=1 en ?AF=0. In het programma is in regel 90 en 310 tevens een anti-dender programmaatje opgenomen.
Het hardware-schema vindt u hieronder. Equivalenten voor het IC 7438 zijn SN5432, SN54LS38, SN7438 en elke andere positieve NAND. Overigens is het boek (6502 Software design: L.J.Scanlon) een uitstekend 6502 boek, dat ook nog aandacht besteed aan de 6522 VIA-chip. tek. 1

TELETYPE-interface. *******
G.Akkermans vereenvoudigde het schema voor de TELETYPE iets. De NAND-poort van de 7438 hoeft n.l. alleen te schakelen tussen wel en geen stroom. Dat kan een enkele transistor ook!! Vandaar deze schakeling. Deze heeft de volgende voordelen: a) hij is goedkoper en b) hij is kleiner, dus makkelijker in te bouwen. Vrijwel ieder type npn-transistor zal in deze schakeling voldoen. tek. 2



tek. 1



tek. 2

Grafisch printen GP-80 printer. "GP 80 H.R." *****
Grafisch printen met de GP-80 van Seikosha, hiervoor moet data bit 8 van de printer aan PB 0 van IC 6522 zitten. Het programma piept als de printer niet aanstaat. Voor het eigen programma moet men vanaf regel 33 het eigen programma intypen. Sluit dit dan af met LINK LL9. Het programma werkt met de volgende codes: S=stop, N=print normaal, I=print invers (=wit op het scherm wordt zwart op de printer en v.v.). De auteur is Hans Rieuwerts.

Enkel of dubbel printen. "ENKEL OF DUB." *********
Het in enkele of dubbele grootte uitprinten van GRAFIC's door Varkevisser uit Katwijk. Toelichting op het programma.

230 ≠AC wordt gebruikt als vlag
240 indien ≠AC=0 dan enkel dumpen
250 indien ≠AC=1 dan dubbel dumpen
280 X-coordinaat
290 Y-coordinaat
300 hulpvlag bij dubbel dumpen
310 byte naar printer
320 teller 1-3 of 1-7
330 vlag bij dubbel dumpen
340 indien 0 dan zijn 3 bits dubbel en 1 enkel aan het printen, indien 1 dan net andersom
540 ≠AB telt in het geval van dubbel dumpen van 1-2
650 als ≠AC=1 dubbel dumpen dus dan wordt het byte verder afgehandeld in PP6-PP16 PP8-PP15 zijn specifiek voor het dubbel printen
1070 vlag, zie boven, is dus 1 of 0
1280 deze subroutine is een trucje om te kijken of een bit op plek (X,Y) aanstaat of niet

Grafisch printen met SEIKOSHA of andere. "GDUMP TOORMAN" ***
Grafisch printen met een 'CENTRONIC's' printer. Met deze printers kan grafisch geprint worden wanneer DATA 8 van de printer niet aan logisch 0 wordt gelegd, maar aan de (te programmeren) bit 0 van poort B. Voor dit programmeren heeft men dan een snel programma nodig. Toelichting bij dit programma:

180 haal startadres objectcode, voorkom overschrijving sourceprogramma
210 initieer variabelen, fill array, enter assembly loop
220 open printer channel, initieer PIA 6522, zet B-poort op output
230 initieer start- en eindlocatie
240-250 initieer printer
260-300 haal 7-bits; stuur deze naar de printer; verschuif masker tot bit 0 aan de beurt is geweest, incrementeer blok (6x7x32 dots) en ga door tot het gehele VDU beeld geprint is, of onderbroken. Zet printer terug in de tekst-mode en schakel het printer-kanaal uit.
360 plaats de tabel met maskers (8 bytes);eind. Ad Toorman

Grafisch printen met de MICROLINE 80. "MICRO 80" *********
De printer wordt eerst in de 132 char/regel en 8 regels/inch gezet en kopieert daarna 'CLEAR 4'. Het is geschreven volgens de standaardisatie norm bit 3 van 8255. Het programma is principieel anders dan dat van bv. Riewerts. Het werkt waarschijnlijk ook voor de EPSON. De lengte van het programma is ≠FE bytes lang. Nadat de printer uitgerateld is, moet nog een Cntr F gegeven worden, omdat het beeldscherm ge'disabled' is. Dit kan ook automatisch door op regel 965 JSRF;LDA&6 bij te tikken. Als na de piep een I wordt ingetikt, wordt het hele beeldscherm geinverteerd en hoor je weer een piep. Drukt men nu op een willekeurige toets dan wordt datgene dat wit is op het scherm zwart op de 'print-out'. Aldus Leendert Bijnagte.

STANDAARDISATIE Printer- en interface schakelingen. xxxxx

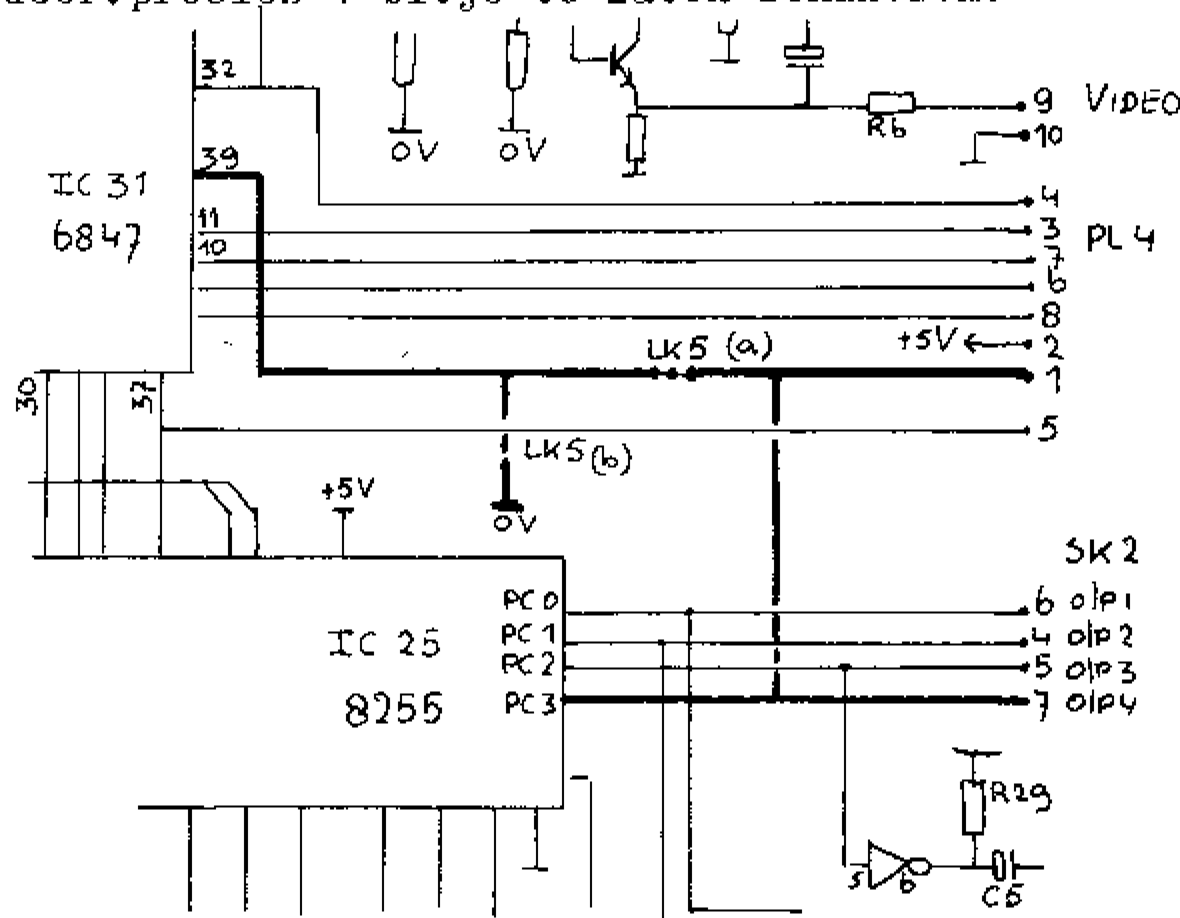
In de paar maanden dat de Acorn Computerclub bestaat, ontvingen wij van de leden meerdere programma's t.b.v. grafisch printen en morse/rtty interfacing. Deze programma's werden, alnaargelang de keuze van de inzender, geschreven hetzij naar bit 0 van de B-poort van de VIA 6522, hetzij naar bitje 3 (dus vierde bit) van de C-poort van de 8255. Als we zo doorgaan, blijven we aan het omsolderen en om-programmeren, wat niet de bedoeling is. Na overleg en bestudering heeft het bestuur besloten om voor beide gevallen (grafisch printen en CW-interfacing) het gebruik van bitje 3 van de C-poort van de 8255 te standaardiseren. Technisch houdt dit in:

A) dat LINK 5(a) wordt doorgekrast
B) dat LINK 5(b) wordt gesloten
C) pin 9 van de printerconnector gaat naar pin 1 PL4

Wat betreft de punten A en B (het resp. doorkrassen en doorverbinden): hierdoor wordt het VDU-beeld bij grafisch uitprinten van het scherm rustiger. Het kan zijn dat u dit verschil niet merkt, maar het is beter om het wel te doen en onvoorziene complicaties te vermijden. Na de handelingen A en B behoudt men de 4 grondkleuren.

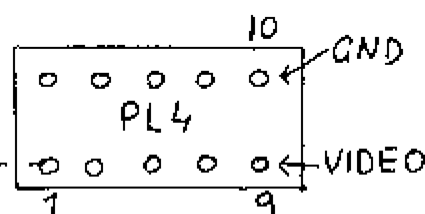
Voor programmeurs: Bitje C3 is normaal LAAG. Met DATA 8 aan bitje C3 ligt de printer dus met data 8 normaal aan 0 d.w.z. TEKST. Poort C bevat 'normaal' D7. Bitje C3 gaat HOOG als we hiervan D8 maken. In basic: ?≠B002=8 in mach.taal: INC≠B002 (en DEC v.v.). De C-poort staat namelijk óók reeds normaal op OUTPUT.

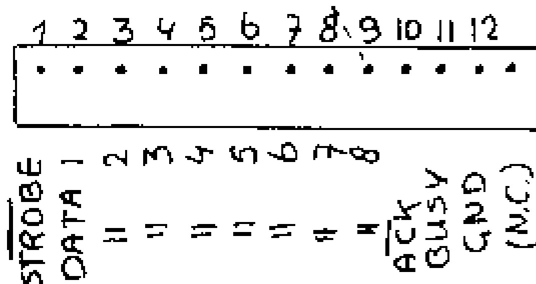
De filosofie achter dit verhaal: we dienen niet een gehele machtige poort van de VIA 6522 te sperren door deze precies 1 bitje te laten schakelen.



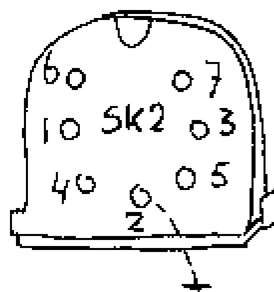
Gedeelte van het Acorn Atom schema. De betreffende verbindingen zijn voor de duidelijkheid iets verdikt.

PL4 gezien op componentenzijde SIDE 2





Chassis-deel printerconnector, gezien aan achterzijde SIDE 2



Achteraanzicht chassisdeel DIN stekker
Signaal CW → 7
tegen 2 = massa
zie Atom schema
Kies: piep = laag
rust = hoog

ADRESSEN met hun beschrijving uit FXXX enCXXX

C278 File shut routine.(RTS)
C2B6 Execute new commando. Blijft onveranderd; 0 wordt met 8 geinitialiseerd.
C2CF Interpreteer Basic string op adres (5,6)
C2D5 Als C2CF; echter nu vanaf adres 100. (commando line buffer)
C3CB Haalt basic accu 16,X enz. en plaatst deze in 52-55
C3CD Als C3CB, echter verplaatst accu naar 0,Y-3,Y
C424 Check of FP-ROM aanwezig is, indien aanwezig wordt de carry geset, zoniet gereset.
C46A Converteert decimale string aangewezen door de basic pointer en getermineerd door een niet ASCII 0 tot 9 char. naar z'n hex equivalent en plaatst deze vanaf 52 in de zero-page, en in de basic accu's 16,X etc. Indien er een variabele wordt ingelezen.
C4E4 Verplaats de basic interpreter pointer naar start eerst volgend statement.
C4F6 Past basic pointer 5,6 en de offset Y(= ZP loc3) zodanig aan dat Y=1 (en ZP loc.3=1)
C558 Roept eerst C4E4 aan; gaat daarna het statement interpreteren waar de basicpointer ((5),Y ,waarbij Y in 3 gesaved is) het statement aanwijst.
C589 Converteert basic accu (16-25-34-43) naar een dec.string die hierna via OSWRCH weggestuurd wordt.
C78B Interpreteert expressie op adres (5,6), resultaat in accu 0 (16-25-34-43)
C8C5 Maak two's compliment van basic accu 16,X enz.
C99D Verplaatst rnd (8-D) naar basic accu 16,X enz.
C9A1 Verplaatst 0,Y-3,Y naar basic accu 16,X enz.
C9D8 Error handler: haalt PC van stack (niet meer nodig) en vervolgt met C9DA
C9DA Error message; accu = error nummer; plaatst inhoud van 10,11 in 5,6 en interpreteerd de string op adres (10,11) hierna door een jump naar C2F2. Deze string staat normaal op adres C9E7 en kan event. door de gebruiker veranderd worden door 10,11 te veranderen.
CA4C Print ASCII karakter in accu, verhoog/verlaag count
CD0F Text input routine. Staat invoer van een string van max. 64 kar.toe. Bij entry wordt de inhoud van de accu als ASCII chr. weergegeven. (Basic input ; accu=3F) de ingevoerde string wordt vanaf adres 100 in het geheugen gezet.
CD0B Als CD0F, echter nu wordt de string vanaf adres 140 in het geheugen gezet.
CD54 Print CR+LF, reset count op 0
CD9B Execute end command; komt terug in basic line input (Commando mode)
CE86 Basic autostart adres. Veronderstelt dat de pointer voor dimensionering arrays op z'n juiste waarde gezet is (normaal:35,36=D,E) evenals TS pointer 12

F668 Decrement 5A,X en 5B,X als zijnde 2 byte teller
F671 Increment 5A,X en 5B,X als zijnde 2 byte teller
F6E2 Point plot routine voor graphics mode 0.
F73B Idem mode 1
F754 Idem mode 2
F76D Idem mode 3
F7AA Idem mode 4
F7D1 Print string beginnend op PC+3 totdat een byte met B7=1 gevonden wordt; nadat geprint is wordt de PC gezet op de instructie met B7=1 (Terminator)

F7F1 Print de hex inhoud van X+1 en X als ASCII kar.
F7FD Print een spatie
F802 Print de hex inhoud van de accu als 2 ASCII kar.
F80B Print accu low nibble als ASCII kar.
F876 Verhoogt Y totdat in de command line (100,Y) een niet space kar. gevonden is.
F87E Onderzoekt ASCII kar. in accu op hex 0-F kar. wanneer een niet hex kar.: set carry; anders reset carry
F893 Haalt hex ASCII data van 100,Y en plaatst deze in 0,X en 1,X. Bij exit bevat A het aantal gelezen ASCII kar., carry is geset
F96E Laad cassette file. De volgende data moet dan aanwezig zijn in ZP: 0,X 1,X adres naam string; 2,X 3,X default laad adres; 4,X positief indien het geen autorun zijn moet.
FA08 Incrementeer 0,X en 1,X als zijnde 2 byte teller. Vergelijk hierna teller 0,X met 2,X; als gelijk Z=1
F8EF Interpreteer string startend op adres 100 als COS commando; alle "*" commando's zijn toegestaan.
FAE5 Save file routine. De volgende data moet in ZP aanwezig zijn:
0,X 1,X adres start van de naam string
2,X 3,X herlaad adres (voor *LOAD en *RUN)
4,X 5,X start executie adres (*RUN)
6,X 7,X adres eerste byte van file
8,X 9,X adres laatste databyte file +1
FB83 Delay loop. Delaytijd wordt bepaald door de waarde in het X reg. * 1/60 sec.
FC38 Print "PLAY TAPE" als C=1; als C=0 "RECORD TAPE" en wacht op toets. Exit met A=13,X en Y vernietigd.
FC40 Print "REWIND TAPE", verder gelijk aan FC38
FC7C Save byte in accu op tape. Checksum in DC wordt verhoogd.
FCEA De eigenlijke print karakter routine. Reageert niet op control kar. en gebruikt alle uP registers.
FD0B Clear B7 Zero-Page byte E0 (P.$6)
FD11 Maakt B7 Zero-Page byte 1 (neg.byte: P.$21) Indien C=1 bij entry; zoniet wordt B7 weer hoog (1)
FD1A Bleep (P.$7)
FD44 Inverteer bit 7 van het kar. waar de cursor op "staat" (EOR met E1)
FD69 Clear screen, home cursor (P.$12)
FD92 Maakt Zero-Page byte E6 negatief (P.$14)
FE52 Print ASCII kar. op beeldscherm en/of printer (mits deze geenabled zijn d.m.v. $6 of $2) alle reg. zijn onveranderd bij exit; controlkarakters worden via een tabel geserviced.
FE66 Wacht een frame (exit als flyback=1)
FE6B Wacht op laag-hoog overgang flyback
FE71 Voer een keyboardscan uit; Als toets ingedrukt: C=0; als geen toets ingedrukt: C=1
FE94 Keyboard input routine. Wacht op toets, ASCII waarde bij exit in accu, anders registers onveranderd.
FEFB Printer routine. Print ASCII kar. in accu, mits geenabled. (Accepteerd en serviced $2 en $3 ook)
FF3F Reset vector wijst naar deze locatie; start met initialisatie operating system vectoren.

BLOCK 0 RAM

0000 Error nummer (Atom Th & Pr)
0001 } Bevat BASIC regelnr. welke
0002 } wordt uitgevoerd
0003 Pointer naar huidig kar. in
0004 Basic progr. in uitvoering

00E6 + Page-mode on (=aantal lege regels)
- Page mode off
00E7 Als bit 5 én 6 1 zijn, 'LOCK' aan
00E8 } Werkruimte
00E9 } v, diversen

0004 Tijdelijke opslag X-Reg.

0005 Pointer naar huidig basic
0006 statement in uitvoering

0007 Print positie tvb. COUNT

0008 Startgetal voor de RANDOM
000C generator (ATP)

000D Systeem variabele TOP
000E " " "

000F Tijdelijke opslag Y-Reg.

0010 Adres van Basic-Error-Handler
0011 normaal is dit ≠C9E7 (ATP)

0012Tekstruimtepointer. NaBRK =≠29

0013 Aantal 'geneste' do-loops

0014 Aantal 'geneste' subroutines

0015 Aantal 'geneste' for-loops

0016 Basic werkruimte:
0058 16,25,34,43= 4byte

0059 Hulpreg.v.Expressie evaluatie

005A X-coördinaat bij PLOTTEN
005B

005C
005D Y-coördinaat bij PLOTTEN

005E PLOT-PARAMETER: 0=MOVE,1=SET, 2=INVERT, rest =CLEAR (unset)

005F Hulpreg. voor
0060 PLOT-routines

0061
006F Basic werkruimte

0070 Floating-point
007F Werkruimte

0080
00DB COS werkruimte

00DC CHECKSUM bij "SAVE" en "LOAD"

00DD Parameter voor FLOAD (bit 7=1) +=tel bloknr. -=+FLOAD

00DE
00DF Adres van de CURSOR

00E0 Positie v/d CURSOR op regel <≠20

00E1 Habitus v/d CURSOR,≠00=uit, ≠80=blokje,kan ook wat anders in

00E2 Werkruimte, bv. voor BACK-UP
00E5 van reg.

00EA Als 'niet 0' geldt 'NOMON'

00EB COS werkruimte
00FD

00FE Normaal 0A='Line-Feed'

0100 Input Line Buffer,
013F normaal ≠40 lang

0140 Behandelruimte voor
017F STRINGS

0180
01FF Voor 6502 STACK (ATP)

0200 NMI routine adres
0201 (niet v. COS of BASIC)

0202 BRK routine adres
0203 normaal C9D8

0204 IRQ routine adres
0205 normaal A000 =IC 24

0206-7 Interpreter-routine Norm.F8EF

0208-9 WRITE CH rout. Norm.FE52

020A-B READ CH rout. Norm.FE94

020C-D LOAD FILE rout.Norm.F96E

020E-F SAVE FILE rout.Norm.FAE5

0210-11 RDR VEC rout. Bij BRK n.C2AC

0212-13 STR VEC rout.Bij BRK n. C2AC

0214-15 Get byte from tape.Nor.FBEE

0216-17 Put byte to tape.Norm.FC7C

0218-19 Print Tape Byte.Norm.FC38

021A-B SHUT FILE rout.adr.Norm.C278 bij RTS

021C
023F Ongebruikt

0240-02E4 BASIC-werkruimte

02EB-0320 ARRAY-adressen @@ naar 2EB en 306, AA naar 2EC en 307 enz.

0321 'Apestaart'=aantal spaties.N=8

0322-033B Low byte v. variabelen A-Z

033D-0356 2de byte

0358-0371 3e byte

0373-038C Hign byte

038D-03C0 Label-adressen, a in 38D en 38E, b in enz.

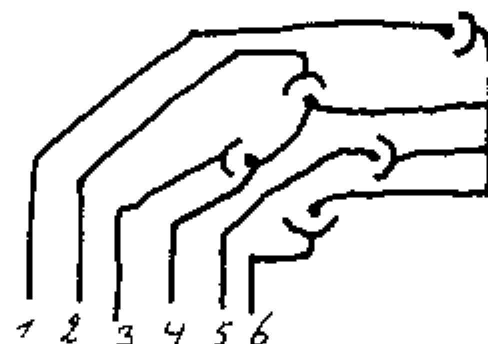
03C1-03FD Basic-werkruimte

03FE-03FF Adressen v.PLOT-POINT rout. F6E2 v. MODE 0;F738,F754, F76D,F7AA resp.v.MODE1,2,3,4

Atari vreugde-knuppeltje:

XXXX

In het Atari vreugde-knuppeltje zitten 5 klikkende microschakelaartjes en er komt een 6 aderig kabeltje uit. De printsporen, zie schema, lopen eigenaardig, doch laten zich door doorkrassen naar believen wijzigen. Voor dezelfde prijs (± f40,-) is een draaiknopdinges te koop. In het volgende nr. meer over de joy-stick.



XXXXX

Knuppel aan de Acorn:
S.v.Eldik kocht, n.a.v. het artikel over de Atari-knuppel aan de Acorn er een. Hij soldeerde direct op de print. Om alles te controleren werd de volgende test (van G.B.) ingevoerd:

```
10 P.?≠B001      Er verschenen 4 keurige rijen met waardes, die veranderden
20 GOTO 10       toen hij aan de knuppel kwam. Geen aanraking dan verscheen
                 er 255. Het ging allemaal een beetje te vlug, dus:
10 P.$12;?≠E1=0     scherm schoon, cursor weg
20 P.?≠B001,$30     inhoud van B001 op scherm en cursor terug naar begin
30 GOTO 20
```

Er verschijnt nu slechts 1 getal op het scherm, dat pas veranderd als er aan de knuppel geduwd wordt. De getallen zijn leuk, maar wat doen we ermee? Voor spelletjes is onze knuppel uitermate geschikt en gemakkelijk te programmeren. We laten het programma steeds naar B001 kijken en reageren op het getal dat daar aanwezig is. Bij het programma 'Moonlander' uit Atom Magic. verving hij de volgende regels voor gebruik joy-stick.

```
20 DIM P(-1);P.$21                   20 Z=≠B001
30[;JSR≠FE71;STY≠80;RTS;];P.$6       vervallen
240 IF R=53 V=V-5;F=F-5;?N=≠D7       240 IF ?Z=239 V=V-5;F=F-5;?N=≠D7
250 IF R=50 H=H-3;F=F-3;?N=≠D7       250 IF ?Z=253 H=H-3;F=F-3;?N=≠D7
260 IF R=38.H=H+3;F=F-3;?N=≠D7       260 IF ?Z=247 H=H+3;F=F-3;?N=≠D7
499e?≠E1=≠60;END                     499eLINK≠FFE3
                                     500 IF ?Z=254;RUN
                                     510 GOTOe
```

Het programma is het zelfde,alleen nu voor de knuppel.

"DEMO JOY" XXXXX

Joy-stick
Demonstratie program tekenen met joy-stick, van Borghaerts. Verklaring: Regel 20 begin maar ergens, 35 zet beginpunt, 50 onderdruk storing, 60 zet inhoud ≠B001 in A, 78 als A=even begin opnieuw. Opmerking: In mode 4 moeten de 4 hoogste bits van B000 'F' blijven.

"JOY STICK" XXXXXXX

Joy-stick II
Dit programma voor het tekenen met joy.stick heeft de volgende voordelen t.o.v. het program uit nr.5:
1)Als de stick heen en weer bewogen wordt, wordt er niet direct een lijn getrokken, maar gaat er slechts een punt heen en weer; tekenen geschiedt na het indrukken van de knop.
2)Tekeningen kunnen worden opgeslagen op band, na teruglezen werkt het program nog en de tekening kan eventueel nog verbeterd worden.
De tekening die we op het scherm willen hebben, tekenen we op een doorzichtig stuk plastic, wat we daarna op het scherm bevestigen. We bewegen de stip nu naar een lijn, die na het indrukken van de knop overgetrokken kan worden. (Schuine lijnen door bv. omhoog-rechts-omhoog-rechts-enz.)
Als we tevreden zijn, kunnen we hem opslaan. Na het intypen van S kunnen

we de recorder starten, en de spatiebalk indrukken. Terug roepen van een tekening geschiedt na het indrukken van L. (Deze handelingen worden verricht zonder naam, dit is veel sneller, wel zelf bijhouden waar het staat op het bandje. Anders regel 210 en 240 aanpassen.) Regel 170 regelt de snelheid van lijntrekken, I=1 TO 100 is erg praktisch. A. de Bruin.

Joy-stick III "JOTTING" ********

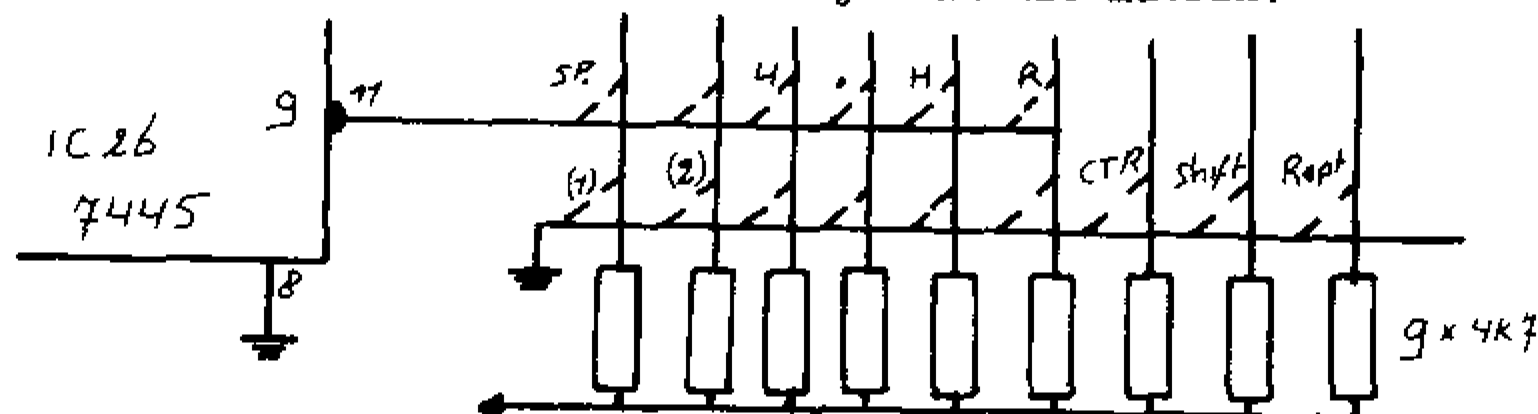
Dit programma heb ik geschreven om, d.m.v. het ons allen bekende 'lolstokje' tekeningen op het scherm te kunnen maken. Iedere keer wordt er in ≠B001 gekeken wat daar voor een getal staat. Bekijken we nu de volgende regel: IF ?≠B001=247; X=X+1 Als we het knuppeltje naar rechts drukken dan wordt er '1' bij X opgeteld. Dat geldt ook zo voor de andere richtingen. Schuin plotten doet hij ook. Stel dat we nu echter iets 'ontplotten' willen, wel dat is heel erg eenvoudig. U drukt gewoon, als toegift op het bv. naar rechts drukken van het knuppeltje ook nog even het drukknopje in. Gaat het plotten u te snel, verander dan de plotvertragingsfactor, bv. in 1000. Aldus: Wim Schreuder.

Pretpook-aflees-routine "PRETPOOK" *********

G.Poot maakte een programma dat de pretpook afleest.

ATARI-knuppeltje aan de ACORN ATOM: door G.Borghaerts *****

Ook hier weer: voor de Acorn Atom nauwelijks een probleem!
Op mijn suggestie in Acorn Nieuws 4 om aan de Atom op de simpelste manier een (digitaal) knuppeltje te hangen, n.l. aan het toetsenbord, kwam een snelle reactie binnen van A.Millenaar. Hij wees op een aantal 'vrije' contactmogelijkheden in de matrix voor het toetsenbord. Helemaal 'vrij' blijken ze niet te zijn, maar wel alleszins bruikbaar en vooral snel! Tekenen we hieronder het betreffende stukje van die matrix:



De weerstanden R17,18,19,20,21,22,23,24 zijn 'boven verbonden met B-poort bit 0,1,2,3,4,5,6,7. van IC 8255. Het zijn 'pull-up' weerstanden, die de ingangen van poort B omhoog trekken naar 5 volt zolang ze niet via een toetscontact én via een van de lijnen van IC 7445 worden omlaaggetrokken en uitgelezen. Volgens het matrix-schema kun je ze ook buiten IC 7445 om, omlaag trekken. De toetsen: Ctr.,Shift en Rept. doen dit ook (Rept. zit echter niet aan de B-poort van IC 8255 maar aan de C-poort bit 6). De toestand van de B-poort van IC 8255 is uit te lezen op de geheugenlocatie ≠B001. Zijn alle bits van deze poort 'hoog', dan staat op deze locatie het getal 255 (≠FF). Trekken we nu bit B0 naar 0 volt, d.m.v. een schakelaar op plaats (1) dan verandert de inhoud van geheugenlocatie ≠B001 in 254. Trekken we bit B1 naar 0 volt d.m.v. een schakelaar op plaats (2) dan verandert de inhoud in 253.
Bit 0 naar 0 volt betekent dus 1 punt eraf
Bit 1 naar 0 volt betekent 2 punten eraf en vervolgens:
Bit 2 naar 0 volt betekent 4 punten eraf, bit 3 naar 0 volt 8 punten eraf, bit 4 naar 0 volt 16 punten eraf, bit 5 naar 0 volt 32 punten eraf, bit 6 naar 0 volt 64 punten eraf en bit 7 naar 0 volt 128 punten eraf.
Door combineren van schakelaars kunnen we zodoende op locatie ≠B001 elk

getal zetten, dat we willen tussen 0 en 255.
Zonder te solderen of zelfs maar de kast open te maken, kunnen we de voor-omschreven gang van zaken controleren.

We typen hiertoe in:

```
10 P.?≠B001     (print inhoud ≠B001)
20 G.10         (blijf dit doen)
RUN (ret)
```

Op het scherm verschijnen nu continu rijen van het getal 255 (≠FF). Drukken we nu op de CONTROLTOETS, dan verandert dit getal in 191. Dat is dus 64 punten minder. Drukken we nu op de SHIFT-toets, dan verandert 255 in 127, dat is dus 128 punten minder. Drukken we op CONTROL en SHIFT tegelijk, dan verandert de 255 in 63, dat is dus 255-64-128.
Omgekeerd is het zo, dat als op locatie ≠B001 het getal 63 staat, dit alleen kan zijn veroorzaakt doordat én Shift én Control zijn ingedrukt. Een andere manier om op 63 uit te komen is er niet! Reken maar na. Zouden we alle schakelaars boven R17 t/m R24 tegelijk indrukken, dan komt in locatie ≠B001 het getal 0. In de praktijk bereiken we op deze manier echter nooit 0. Dit komt omdat één van deplaatsen voor ons doel onbruikbaar is, n.l. de plaats boven R22. Het laagste bereikbare getal is dus 32. De reden hiervan is de volgende. De laatste handeling bij het intypen van het programma'tje (zie boven) was het indrukken van de returntoets. Nu is het zo, dat na elke reset (break) of returntoets de lijn nr.0 aan pin 1 van IC 7445 naar 0 gaat. Via deze lijn trekt het indrukken van de ESC-toets de bit B5 van IC 8255 naar 0, wat dan weer een lopend program onderbreekt en de cursor teruggeeft. Precies hetzelfde gebeurt als we boven R22 een schakelaar aanbrengen die eveneens bit 5 naar 0 zou trekken. Een schakelaar op die plaats zou dus elk spel onderbreken, wat niet de bedoeling is.

Als bruikbaar houden we dus over 7 schakelplaatsen n.l. S17 (boven R17), S18,S19,S20,S21,S23 en S24 waarvan elk afzonderlijk of gecombineerd gebruikt, een eenduidig getal geeft op geheugenlocatie ≠B001, direct uitleesbaar.
Makkelijker kan het al niet. En sneller óók niet, want schakelen op deze plaatsen heeft voorrang doordat het in feite buiten de matrix om gaat. Dit is dan ook de reden dan Control, Shift en Rept. dáár schakelen.

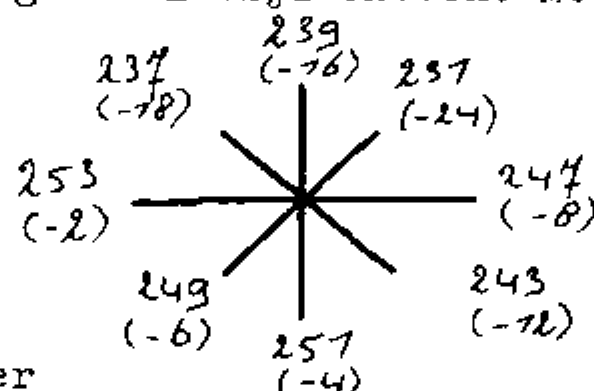
Omdat de schakelaars uiteraard niet in de matrix worden gesoldeerd, maar buiten de kast in de joy-stick, moeten we -het beste aan de weerstandjes-draadjes solderen en verbinden met een stekkerbus aan de zijkant van de kast. Voor dit doel zijn schitterende stekkers met chassis-deel te koop van het merk HIRSCHMAN type MEB-10 (10 polig).
Beginnen we nu te onderzoeken aan het andere einde van het verhaal, dan blijkt de ATARI-joy-stick bijzonder goed in onze opzet te passen. In Acorn Nieuws 4 werd een schema'tje getekend van het printje in deze joy-stick. Er kunnen 1,2 of 3 schakelaars tegelijk worden ingedrukt, bv. DRUKKNOP + BOVEN - RECHTS. Er zitten dus 5 schakelaars aan, boven, onder, rechts, links en drukknop. Met die 5 schakelaars ontstaan 9 mogelijkheden, n.l. 8 richtingen + drukknop.

Van de 7 mogelijke schakelplaatsen in de matrix moeten we er nu 5 kiezen. Er zit niet het minste verschil in. Kiezen we derhalve maar voor de vuist weg S17,S18,S19,S20 en S21. Op het toetsenbord houden we dan nog Shift en Control over.
Ook in de joy-stick maakt het technisch niets uit welke poort-lijn aan welke schakelaar komt. Kiezen we ook maar voor de vuist weg: de DRUKKNOP wordt S17 LINKS wordt S18 ONDER wordt S19 RECHTS wordt S20 en BOVEN wordt S21.

Op deze wijze heb ik een ATARI-knuppeltje aan de Atom gesoldeerd en we krijgen dan op de locatie ≠B001 de volgende getallen:

a) Zonder ingedrukte drukknop is het getal altijd oneven. Met ingedrukte drukknop altijd even.

b) Zonder drukknop geven de acht richtingen van de joystick de volgende getallen. Mét drukknop alles één punt minder:



Men krijgt deze getallen bij deze richtingen wanneer van het ATARI-snoer de oranje draad verbindt met R17, de groene draad verbindt met R18 de blauwe draad verbindt met R19, de bruine draad verbindt met R20 de witte draad verbindt met R21 en de zwarte draad met 0 volt van het board.

Bij een ander knuppeltje dan van ATARI kunnen de draadkleuren die bij bepaalde schakelaars horen wellicht anders zijn. Dat doet er niet toe als we er maar voor zorgen, dat bij de drukknop en bij de richtingen van de knuppel dezelfde getallen ontstaand als volgens het tekeningetje.

!! Wanneer we in de club de zaken op deze wijze standariseren wordt het mogelijk om bestaande programma's naar deze joy-stick om te schrijven en ervoor te ontwerpen en deze in het archief op te nemen!!!

Nog één praktische hint: Let er vooral op, dat de draadjes aan de goede kant van de weerstandjes gesoldeerd zitten. De verkeerde kant is, waar ze allemaal aan de +5 volt zitten. Zou kortsluiting geven!

Met het ATARI-knuppeltje aangesloten, zoals hiervoor omschreven, kunnen we reeds meer op het gebied van behendigheidsspelen als momenteel mogelijk is in bestaande spelen. We kunnen bv. een keeper zijn doel laten verdedigen tegen random afgeschoten ballen. In 8 richtingen kan de doelman op de bal toelopen, dan stoppen, dan drukknop ín en dan de bal in 8 te kiezen richtingen wegschieten. Enz. programmeurs kunnen hun gang gaan!

Het aansluiten van twéé knuppeltjes kan principieel op dezelfde manier en aan dezelfde schakelaars. Dezelfde leidingen vanaf de weerstanden R17 t/m R21 moeten dan óók naar een 2e stekkerbus en daar komt de 2e joy-stick aan. Alleen met de retour-leiding van de 2 joy-sticks moet u verschillende dingen doen.
In het verhaal over de 1e joy-stick hebben we de retourleiding vanaf de joy-stick (zwart) aan 0 (Aarde) van het board gelegd. We kunnen die retourleiding echter óók verbónden met pin 1 van IC 7445, dat is de bovenste horizontale leiding van de toets-matrix. Let op: die lijn zelf heet 0. Ons knuppeltje werkt dan even goed zolang IC 26 die lijn 0 maar aan 'aarde' houdt. We hebben gezien dat dit het geval is na elke 'reset', 'break','run' enz. Op elk moment binnen een programma kunnen we die lijn 0 naar 'aarde' brengen door (in basic) 10 ?≠B000=0.
We kunnen dit zonder enig soldeerwerk controleren door in te typen:

```
10 ?≠B000=0        (leg IC 26 lijn 0 aan 0)
20 P.?≠B001        (lees inhoud loc.B001)
30 G.20            (blijf dit doen)
RUN
```

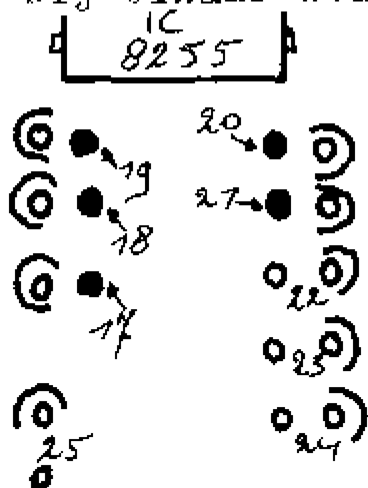
Drukken we op toets 3, dan verandert de 255 in het getal 253. Dat is het zelfde, wat de joystick doet bij naar links bewegen.

Typen we nu in:

```
10 ?≠B000=1     (leg IC 26 lijn 1 aan 0)
20 P.?≠B001     (lees inhoud loc.B001)
30 G.20
```

We zien weer reeksen van 255. Toets 3 doet nu niet meer. In de plaats daarvan krijgen we nu het getal 253 door indrukken van de toets nr. 2. Deze toets zit op dezelfde plaats als 3 18 maar nu met de retourleiding naar de lijn nr 1 van IC 26, dat is pin 2.

Het practische soldeerwerk:
Als we de computer op z'n kop leggen met de toetsen naar ons toe en we schroeven de bodem eraf, dan zien we de weerstandjes R17 t/m R21 in een groepje bij elkaar staan. Kies voor de aansluitingen de in hiernaast staande tekening zwart gemaakte aansluitpunten. Het beste en het netste gaat dit aan de ANDERE kant van het board.



Dit zijn dus de 5 signaaldraden S17,S18,S19, S20 en S21 aan de weerstanden R17,R18,R19, R20 en R21.
De andere kant van alle weerstanden zit aan +5 volt.
Mooi werk krijgt u door e.e.a. uit te voeren met een stuk zg. regenboog flat-kabel.

Houden we rekening met de toekomst en gaan we nu reeds zo 'universeel' mogelijk te werk, dan gebruiken we deze 10 aderige regenboogkabel en de 10 pen's HIRSCHMAN stekkers. Van deze kabel verbinden we draad 1 t/m 5 (bruin,rood,oranje,geel,groen) aan de stekkerpennen 1,2,3,4,5 en aan resp. R17,18,19,20,21. Daarna de draden 8,9,10 (grijs,wit,zwart) aan resp. pin 2 van IC 7445 (26), pin 1 van IC 7445 en zwart aan 'aarde' van het board. Aan de stekkerkant komen deze 3 draden dan aan de middelste 3 pennen van de stekker, dus de pennen 8,9 en 10. Er zijn dan 2 draden én 2 stekkerpennen over waarmee u later naar wens retourleidingen (voor meer joy-sticks) of meer schakelfuncties per joy-stick wilt uitvoeren.

De retourdraad naar 'aarde' gebruikt u als u alleen maar programma's gebruikt voor één joy-stick. Zo aangesloten werkt het program het snelste. Gebruikt u echter de draad wit als retourleiding naar pin 1 van IC 7445 dan werkt zo'n programma zonder enige wijziging. Het 'uitlezen' gebeurt dan echter een fractie van een sec. trager en even snel als het uitlezen van de 2e joy-stick, waarvan de retourleiding via de grijze draad naar pin 2 van IC 7445 gaat. Bij twee (digitale) joy-sticks blijft dus de zwarte draad naar 'aarde' ongebruikt. We kunnen deze dan evt. later gebruiken als massadraad bij aansluiten van 'analoge' joy-sticks tesamen met +5 volt op een van de resterende draden.

STANDAARDAANSLUITING JOY-STICKS aan ACORN-ATOM. April 1982.
G.Borghaerts.

Routines en disassemblers

Inkey faciliteit. ****
Uit Practical Computing 10/81. Een INKEY-faciliteit die met een enkele Basic lijn te verkrijgen is:
X=≠81;?X=≠20;X?1=≠71;X?2=≠FE;X?3=≠84;X?4=≠80;X?5=≠60 Als subroutine aan te roepen met LINK≠81 De toets is te vinden door Peeken op locatie ≠80 of te benoemen met bv. K=?≠80

Input routine. ********
U wilt een overal in te zetten INPUT-routine, die alleen digits accepteert?

```
1 P=≠xxxx;DIMLLO;[;:LLO JSR≠FE94;CMP@≠30;BMI LLO
2 CMP@≠3A;BPL LLO;JSR≠FE52;ADC@≠CF;STA≠90;RTS;];END
```

Aanroepen met LINK≠xxxx, het resultaat vindt u op ≠90. Wilt u het resultaat niet op het scherm, verwijder dan JSR≠FE52

FIND/TOOLBOX aan/uit. "FIND" "TOOLBOX" *******
Twee programma's, van A.Marchal, bedoeld om FIND en TOOLBOX in een programma te gebruiken.

Hexdump met ASCII. "HEXDUMP" ********
Programma, ingezonden door Bram Poot, geeft een hexdump met ASCII karakters. Het is geplaatst op ≠E000 t.b.v. de schakelkaart. Kan ook elders. Het programma kan ge'link't worden met LINK≠EBF9, na toevoeging van de 7 laatste bytes, b.v. met de monitor van Roel Heuvel. ?≠FE=7 i.p.v. 10 bij gebruik van een EPSON-printer.

Automatisch regelnummering. "AUTO REGELNR" *******
Sinds ik de Acorn Atom heb, irriteer ik me er telkens aan dat bij het intikken van programma's ik zelf de nummering moet bijhouden en ingeven. Dat kan de computer zelf veel beter. Het eenvoudige routinetje dat ik daarvoor maakte laadt men in een stuk geheugen met een hoog adres, bv. ≠8200. (Pointer op adres 18 aanpassen, NEW ingeven en laden.) Na 'RUN' vraagt de routine om een tekstgebied. Geef in de eerste 2 posities van het adres, dus ≠29 voor ≠2900 enz. Daarna wordt gevraagd om het eerste regelnummer en de stappen voor verhoging. Bovendien wordt standaard in de eerste regel kommentaar geplaatst (programmanaam). Het 'END' hoeft niet zelf te worden ingetypt. Heeft u alles ingetypt dan is een simpele RETURN genoeg om het programma te stoppen. De routine genereert een 'END', en zet de pointer op adres 18 op het tekstgebied dat u zojuist gevuld hebt. Tik zelf nog even END om de 'TOP'-pointer aan te passen.
J.H.Man

Mirakel. "MIRAKEL" ********
Het programmaatje 'MIRAKEL' is in feite een gewone disassembler die snel en correct functioneert. Deze disassembler maakt -in tegenstelling tot sommige BASIC-programma's- een duidelijk onderscheid tussen absolute en zero-page adressing-mode (neem bv. de instructie LDA≠52,Y. Dit is géén zero-page instructie en wordt dan ook afgedrukt als LDA≠0052,Y).
De uitvoer is in de uitgebreide versie te stoppen en te onderbreken met behulp van de ESC-toets. Tot zover is er eigenlijk geen reden om dit programma de naam 'MIRAKEL' mee te geven. Het programma heeft echter nog een bijzonderheid: in de geassmbleerde basis-versie gebruikt het slechts 243 bytes. Deze waarde is inclusief de data-tabel aan het eind van het programmaatje.
In de basis-versie is het Mirakel een subroutine die volledig zelfstan-

dig kan functioneren en als zodanig een zeer krachtig hulpstuk voor gebruik in basic- of machinetaal programma's. Bv. !V=≠C250;DO LINK LL1; UNTIL 0 hierin is V het adres voor 4 opeenvolgende 'vrije' geheugenplaatsen in zero-page (in dit programma is V=≠70 gekozen).
In de uitgebreide versie wordt het mirakel een complete disassembler die toch maar een minieme 255 bytes aan geheugen gebruikt. De uitgebreide versie ontstaat door toevoeging van: 220:LL0 JSR≠C504;JSR LL1;BMI LL0 en op regel 340 340....JSR≠FA08;BNE LL10;PLA;PLA Het mirakel werkt nu of van een op te geven start-adres tot een te kiezen eind-adres of van het start-adres t/m ≠FFFF. Bv. !V=≠C400C250;LINK LL0 Let hierbij op de omgekeerde volgorde van start en eind-adres, die hier noodzakelijk is omdat ik !V gebruik (! zet de meest linkse byte het laatst in het geheugen). Bv. !V=≠C250;LINK LL0 Als u de uitvoer heeft stil gezet met de ESC-toets, dan kunt u weer verder listen met LINK LL0, !V is dan niet meer nodig.
Het Mirakel wordt hier gepresenteerd, omdat het door de zéér korte lengte bij uitstek geschikt is voor gebruik in EPROM of cmos-RAM op ≠A000 (een nieuw statement misschien?). Zoals het Mirakel is opgenomen hebben we te maken met de uitgereide versie en als u het gerund heeft, verschijnt de tekst 'program uses 255 bytes'.
Voor de mensen die het Mirakel als subroutine willen gebruiken nog de volgende informatie: De instructie-lengte min één staat op locatie ≠0. De adressing-mode staat op ≠F in de volgende code:

0=implied (of illegal)	5=zero-page	A=(indirect),Y
1=accumulator	6=immediate	B=(indirect,X)
2=relatief	7=immediate	C=absolute
3=zero-page,Y	8=absolute,Y	D=(indirect)
4=zero-page,X	9=absolute,X	E=absolute,Y

Opgemerkt kan nog worden dat ≠F niet direkt vanuit basic opgevraagd kan worden en dat het Mirakel geen gebruik maakt van de FP-ROM, waardoor het MIRAKEL mogelijk nog korter zou worden. Door:Frans v. Hoesel.

Pass-Cass Routine. ********

De routine doet wat hieronder is afgebeeld. Ze geeft een assemblerlisting zódanig geformatteerd dat labels op een apart veld worden afgedrukt en niet meer tussen de mnemonics terecht komen. Ikzelf heb deze routine met het commando 'PASS' (print assembler) opgenomen in de jumper-soft. U kunt de routine ook 'los' zelf ergens inbouwen, bv. achter slot op de geheugenkaart, bv. vanaf ≠706A zoals in bijgaand voorbeeld. U roept de routine dan aan met LINK LL0 (in dit geval LINK≠706A en gaat er weer uit met LINK LL10 (in dit geval LINK≠70DE). Dit laat u een programma'tje doen, zoals (in alg.vorm):

```
10 DIM LL10,SS2
20 GOS.z;GOS.z;END
30zP=≠....           (beginadres)
40[
50                   (hiertussen de routine
..                    intypen in
90]                       assemblervorm)
100 RETURN
```

```
 70 706A
 80 706A A9 7D    :LL0  LDA@LL1&≠FF
 80 706C 8D 08 02       STA≠208
 90 706F A9 70          LDA  LL1&≠FFFF/256
 90 7071 8D 09 02       STA≠209
100 7074 A9 00          LDA@0
100 7076 85 91          STA≠91
100 7078 85 92          STA≠92
```

```
100 707A 4C 5B C5           JMP≠C55B    200 70B5 A9 00     :SS1  LDA@0
110 707D 85 90     :LL1     STA≠90      200 70B7 85 91           STA≠91
110 707F C9 0A              CMP@≠0A     200 70B9 E6 92           INC≠92
110 7081 D0 06              BNE LL2     210 70BB A5 92           LDA≠92
120 7083 20 B5 70           JSR SS1     210 70BD C9 41           CMP@≠41
120 7086 4C AB 70           JMP LL5     210 70BF B0 01           BCS LL6
130 7089 A5 07     :LL2     LDA≠07      210 70C1 60              RTS
130* 708B C9 15             CMP@21      230 70C2 A9 0C     :LL6  LDA@≠0C
140* 708F F0 0A             BEQ LL3     240 70C4 20 52 FE        JSR≠FE52
140 7091 C9 1A              CMP@26      240 70C7 A9 00           LDA@0
140 7093 D0 16              BNE LL5     240 70C9 85 92           STA≠92
150 7095 20 CC 70           JSR SS2     240 70CB 60              RTS
150 7098 4C AB 70           JMP LL5     250 70CC A5 91     :SS2  LDA≠91
160 709B A5 90     :LL3     LDA≠90      250 70CE D0 01           BNE LL8
160 709D C9 3A              CMP@≠3A     250 70D0 60              RTS
160 709F F0 10              BEQ LL7     260 70D1 A5 90     :LL8  LDA≠90
170 70A1 A2 06              LDX@6       260 70D3 C9 20           CMP@32
170 70A3 A9 20              LDA@32      260 70D5 D0 01           BNE LL9
180 70A5 20 52 FE  :LL4     JSR≠FE52    260 70D7 60              RTS
180 70A8 CA                 DEX         270 70D8 A9 20     :LL9  LDA@32
180 70A9 D0 FA              BNE LL4     270 70DA 20 52 FE        JSR≠FE52
190 70AB A5 90     :LL5     LDA≠90      270 70DD 60              RTS
190 70AD 20 52 FE           JSR≠FE52    290 70DE A9 52     :LL10 LDA@≠52
190 70B0 60                 RTS         290 70E0 8D 08 02        STA≠208
195 70B1 E6 91     :LL7     INC≠91      300 70E3 A9 FE           LDA@≠FE
195 70B3 D0 F6              BNE LL5     300 70E5 8D 09 02        STA≠209
130* 708D 90 1C             BCC LL5     300 70E8 4C 5B C5        JMP≠C55B
```

Enkele nieuwe routines voor de Monitor R.H. ********

De routines PHXY en PLXY.

Deze routines zetten de inhoud van het X- en Y-register op de stack. Deze routines worden als subroutine aangeroepen en daarom dient enig schuifwerk in de stack te worden verricht. Daarvoor is tijd nodig en wel 101 machine cycles ofwel 101 microsec. (bij 1 Mhz clock frequentie).
De routine PHXY neemt inclusief de aanroep 131 cycles in beslag en gebruikt tijdelijk 1 geheugenplaats om de accumulator te saven. Dit is gedaan om ingewikkeld schuifwerk in de stack te voorkomen. De routine zet het X- en Y-register op de stack en laat daarbij de zowel A, X als Y ongewijzigd.
De routine PLXY neemt inclusief de aanroep 133 cycles in beslag. Het effect van deze routine is tegengesteld aan PHXY. Het op de stack gezette X- en Y-register wordt weer teruggezet en de ACCU blijft ongewijzigd.
Om de SWAP routine als universeel bruikbare routine te veranderen moet er direct achter de BNE SWAP1 een RTS geplaatst worden. Wel worden de X- en Y-registers vernietigd (of eigenlijk alleen de inhoud ervan (!!)). De SWAP routine kan in de toekomst eventueel worden aangepast om een willekeurig aantal data bytes te verschuiven. Hierbij moeten de addressen die nu 106 en 104 zijn variabel worden en toegevoegd een routine die deze adressen berekend. Bij aanroep bevat de Accu het aantal te schuiven bytes en het aanroepadres bepaalt de richting. Op deze manier kan op een eenvoudige manier gegevens tussen een subroutine en het hoofdprogramma worden uitgewisseld, zonder dat er vaste geheugenplaatsen gebruikt hoeven te worden. Het is wel verstandig om de gehele pagina 1 als stack te reserveren, dus de input en stringbewerkings buffers verhuizen naar ergens anders.
Nog een opmerking over de Atom BASIC: deze basic maakt slechts beperkt gebruik van de stack en reset de stackpointer geregeld.

```
SWAP    TSX
        LDA≠02       er moet 2 maal
SWAP1   PHA          geschoven
        LDA≠0106,X
        LDY≠0104,X
        STA≠0104,X
        TYA
        STA≠0106,X
        DEX
        PLA
        SEC
        SBC≠01
        BNE SWAP1
        LDA≠0108,X   restore Y
        TAY
        LDA≠0107,X   restore X
        TAX
        RTS
```

```
PHXY    STA ASAVE    save A
        TYA
        PHA          push Y
        TXA
        PHA          push X
        JSR SWAP
        LDA ASAVE    restore A
        RTS
PLXY    STA ASAVE    save A
        JSR SWAP
        PLA          pull X
        TAX
        PLA          pull Y
        TAY
        LDA ASAVE    restore A
        RTS
```

Roel Heuvel

TOELICHTING OP DE MONITOR. RH.

DEEL1

(bandje 6 in het Clubarchief)

ALGEMEEN:

De monitor gebruikt de dubbele punt (:) als prompt. (I.p.v. de > in BASIC) Bij het verschijnen van de : verwacht de monitor een opdracht. Een opdracht bestaat altijd uit 1 letter of letterteken, dus bv. een L of een M of een + etc. Wil man de opdracht toepassen op een adres, dan typt men na de : eerst dat adres in, gevolgd door de opdracht. Alles zonder spaties. En het ≠ teken. In de navolgende toelichting wordt AAAA gebruikt voor AANVANGSADRES en EEEE voor EINDADRES en DDDD voor BESTEMMINGSADRES (Destinationr).

VOORBEELDEN:

-- L -- De opdracht L geeft 15 regels machinecode-instructies vanaf het opgegeven adres. (Aangenomen natuurlijk, dat op het opgegeven adres een machinetaalprogramma staat, dat u wilt bekijken en/of veranderen.)

```
  :AAAAL (ret)
```

Voorbeeld:

```
  :C2B2L (ret)
C2B2  A9 29    LDA  @≠29
C2B4  85 12    STA  ≠12
C2B6  A9 0D    LDA  @≠0D
```

enz. 15 regels.

(Voor de betekenis van de MNEMONICS LDA, STA enz. zie MANUAL blz. 181 en verder. Zie ook CURSUS ASSEMBLER en ACORNSOFTprogramma PEEKO-COMPUTER.)

De 15e regel van het voorgaande voorbeeld van de L (List) opdracht wordt:

```
C2CC  8D 21 03 STA  ≠0321
:
```

Door nu achter de : alleen L in te typen, volgen de volgende 15 regels. Dit kan men blijven herhalen.

CONCLUSIE: Het commando L geeft een disassembler, pagina na pagina.

-- G -- Met de opdracht G kan men naar een machinetaal programma springen Op de zelfde wijze als dit gebeurt met LINK in BASIC. En net als met LINK start G het machinetaalprogramma. Als voorbeeld kiezen we het machinetaalprogramma OSCRLF met als startadres ≠FFED. (Zie Manual hoofdstuk 25.) Deze routine volgt normaal op het indrukken van de returntoets. Met het aanroepen van deze routine, gevolgd door het indrukken van de returntoets, gebeurt dit dus TWEE maal, Let Op!

Voorbeeld:

```
  :FFEDG (ret)


  :
```

Er zijn dus TWEE regels overgeslagen. Men noemt dit 2 LINEFEEDS.

-- P -- (P van Processor-register.) Met deze opdracht verschijnt de inhoud van alle processor-registers.

```
  :P (ret)
```

Voorbeeld:

```
  :P (ret)
 PC    A  X  Y  S  P  NV BDIZC
C2B4  29 00 05 FB 3D 00110101     (U kunt ook andere waarden krijgen.)
```

Toelichting: De PC op de eerste regel betekent PROGRAM COUNTER. Na uitvoering staat de PC op C2B4. De A staat voor het ACCUMULATOR-REGISTER (normaal ACCU genoemd) en is geladen met ≠29. X en Y zijn INDEX-REGISTERS van de 6502. De S staat voor STACK-POINTER. Een soort 'prikpen' met notities van door de 6502 te onthouden DATA of ADRESSEN. De P staat voor

PROCESSOR-STATUS-REGISTER. Hierin staan de beruchte FLAG's. Te weten N(Negatief), V(Overflow), B(BRK-flag), D(Decimale rekenwijze), I(Interrupt-blokkeerflag), Z(Als de Accu 0 wordt, dan wordt de Z-flag 1), C(De Carry, de '1-onthouden' bij rekenen, u weet wel).
Waar dient deze opdracht voor? Wel: de waarden die u met de P opdracht afgedrukt krijgt zijn niet de werkelijke (momentele) inhouden van de registers, maar copieen die tevoren van deze registers zijn gemaakt. U kunt deze copieen veranderen door NU een dubbele punt in te typen en vervolgens de waarden van de registers zoals u die hebben wilt. De eerste ingetypte waarde komt dan onder A, de tweede onder X enz. Een waarde die niet veranderd moet worden, dient dus opnieuw te worden ingetypt. Let ook op de spaties!
Voorbeeld:

```
  :P (ret)
 PC   A  X  Y  S  P  NV BDIZC
3000  DA 03 A0 EA 31 00110001
::DA FF A0 D5 (ret)
  :P (ret)
 PC   A  X  Y  S  P  NV BDIZC
3000  DA FF A0 D5 31 00110001
:
```

De laatste P-opdracht drukt dus het resultaat van de verandering af. We hebben echter dus de 'copieen' veranderd. De copieen worden in de echte registers teruggeschreven op het moment TUSSEN het geven van de opdracht G, S en T en de uitvoering van deze opdracht door de computer.
-- S -- (van STEP) Met deze opdracht kan men 1 instructie doen uitvoeren van een machinetaalprogramma.

```
  :AAAAS (ret)
```

Voorbeeld:

```
:C2B2S (ret)
C2B2  A9 29     LDA  #29
 PC   A  X  Y  S  P  NV BDIZC
C2B4  29 00 05 FB 3D 00111101                 (U kunt ook iets anders
:                                                krijgen onder X Y en P)
```

Toelichting: De eerste regel kennen we van de L-opdracht, een disassemblerregel. De tweede en derde regel kennen we van de P-opdracht. De 1e regel is de machinecode-instructie, de 2 volgende regels geven de registers weer NA uitvoering van die 1e regel. U ziet, dat de PC is bijgesteld van B2 naar B4 omdat de instructie 2 bytes lang was; de 1e byte A9 (de OPCODE:'laad ACCU met..) het 2e byte 29 (de OPERANT).
-- T -- (van TRACE) Met deze opdracht wandelen we door een machinetaalprogramma om het verloop na te gaan. Dit kan in de PAGINA-MODE wanneer Ctrl-N voor de opdracht wordt ingetoetst, gevolgd door returntoets. (Geeft geen ERROR.) De opdracht T wordt beeindigd door ESC in te drukken en vast te houden tot een volle instructie is afgerond en de dubbele punt weer verschijnt. Let op! Bij 2 keer ESC of dender gaat de monitor terug naar BASIC. De T-opdracht is een automatisch herhalende STEP-opdracht.

DEEL2

Algemeen: Alle monitor opdrachten bestaan uit een letter of symbool, waar een hexadecimaal adres aan vooraf mag gaan. De meeste opdrachten hebben betrekking op een adres en zullen dit adres ook veranderen. Die verandering houdt meestal 'met 1 ophogen' in, elke keer dat de opdracht wordt uitgevoerd. Indien er voor de opdracht geen adres is opgegeven, wordt het default adres gebruikt. Het default adres is het adres dat bij een voorgaande opdracht is achtergelaten. De monitor bewaart meer adressen, elk met een eigen functie. Deze adressen en hun functie zijn in

onderstaande tabel weergegeven.

Program Counter	PC	(90)
Adres 1 laatst weergegeven adres	A1	(92)
Adres 2 laatst ingetypte adres	A2	(94)
Adres 3 volgend te veranderen adres	A3	(96)
Adres 4 bestemmingsadres bij M en V	A4	(98)
Adres 5 gereserveerd	A5	(9A)

In de samenvatting van de mogelijke instructies staan de bij de opdrachten gebruikte adressen tussen (). De adressen A1, A2, A3, A4 en A5 worden door de S en T opdracht als kladblok geheugen gebruikt en hun inhoud gaat dan verloren. De copieen van de processor registers volgen op A5. De 4 bytes na A5 dus niet voor andere zaken gebruiken. Het totale gebruikte Z-page bereik loopt van 0090 tot 00A9.

SAMENVATTING MONITOR opdrachten:

(PC)L............List 15 instructies van een machinetaal programma, beginnend vanaf adres PC.

(PC)S............Step simuleert 1 machinetaal instructie en laat de instructie voor en de processor registers na simulatie zien.

(PC)T............Trace is een doorlopende S-opdracht. Trace is te onderbreken door ESC enige tijd ingedrukt te houden.

(PC)G............Go springt naar een machinetaal programma en zet vooraf de processor registers in de processor.

(A1).(A3)........Geeft een hexdump van adres A1 t/m adres A3, 8 per regel

(A3)(CR).........Geeft de inhoud van adres A3.

(CR).............Geeft de inhoud vanaf het laatst weergegeven adres totdat het adres op 7 of F eindigt. Herhaald intypen van (CR) geeft dus steeds de inhoud van de volgende 8 geheugenplaatsen.

(A4)<(A1).(A3)M..Verplaatst het geheugengebied A1 t/m A3 naar een gebied dat op A4 begint tot aan (A4+(A3-A1)).

(A4)<(A1).(A3)V..Vergelijkt het geheugengebied A1 t/m A3 met het gebied A4 t/m (A4+(A3-A1)) en geeft de verschillen aan.

(A3):(A2) (A2)..Plaatst de ingetypte data (A2L) in achtereenvolgende geheugenplaatsen. Bij veel data op de volgende regel gewoon verder gaan door :(data) (data) (data) (data) (data) etc. in te typen.

(A1)+(A3)........Geeft de som als 1 byte two's complement getal.

(A1)-(A3)........Geeft het verschil als 1 byte two's complement getal.

P................De copieen van de processor reg. worden zichtbaar gemaakt.

Q................Terugkeer naar het aanroep programma. De monitor dient dan wel als subroutine te zijn aangeroepen. Anders ESC.

*COS.............Ieder COS commando achter de * wordt uitgevoerd, zoals beschreven in de Manual.

Enige bruikbare routines: Hieronder volgen enkele bruikbare routines. De adressen die bij deze routines gegeven zijn hebben betrekking op een versie die op adres 7C00 (start) geassembleerd is. (bandje 6 versie en 9).

INDIS 7C53 Geeft 1 regel met een gedisassembleerde instructie, incl. adres en hexbytes, door INDIS1 en INDIS3 uit te voeren.

INDIS1 7C00 Drukt de PC af en vervolgt met INDIS2.

INDIS2 7C0D Bepaalt de lengte, plaats in de mnemonictabel en het print format van de opperand. Deze routine eindigd met RTS.

INDIS3 7C56 Drukt de hexbytes van de instructie af en vervolgt met INDIS4

INDIS4 7C6D Drukt de mnemonic met opperand gegevens af en berekent vervolgens de nieuwe PC. De routine eindigt met RTS.

Tabellen: 7FA2 index naar tabel 7FE6, 7FE6 bevat lengte en adressering,

7FF3 printcode's opperand, F155 mnemonic linkerbyte, F195 mnemonic rechterbyte.

Om een overzicht te krijgen in de werking van deze routines, is een geprinte gedisassembleerde listing van deze routines zeer handig. Denkt u er niet uit te komen dan doet u net of u de 6502 bent en gaat volgens die listing te werk. Gebruik erbij een 6502 Programming Manual. Veel succes met analyseren. Daarbij leert u uw eigen programma's te schrijven. Leg eens een verzameling van universeel toepasbare routines aan. Ze kunnen overal vandaan gesuffeld zijn.

ROEL HEUVEL.

Uitbreiding van de BASIC INTERPRETATOR

"BASIC UITBR" *****

De Atom basicinterpretator is opgebouwd uit een aantal min of meer zelfstandige routines, die een enkel BASIC commando kunnen uitvoeren.
Deze routines worden door de Basic interpretator opgeroepen, als een bep. statement (bv. 'FOR' of 'UNTIL') herkend is m.b.v. een tabel waarin alle voor Basic bekende woorden zijn opgenomen. Deze tabel staat in ROM vanaf adres ≠C000. Deze tabel is, omdat hij in ROM staat dus niet zomaar uit te breiden. Er is echter een leuke truc mogelijk: kan de Basic interpretator een bep. woord niet vinden in de tabel (bv. UNTOL), dan wordt er een software interrupt gegeven, een BRK instructie.
Wat doet deze BRK nu precies?
Allereerst wordt de programcounter van dat moment op de stack gezet, en daarbovenop het statusregister van de 6502. Vervolgens wordt op adres ≠FFFE en ≠FFFF gekeken waar de IRQ/BRK HANDLER staat. In het geval van de Atom wordt er naar adres ≠FFB2 gesprongen. Daar wordt na enig testwerk (zie ook Atomic Theory & Practice) de opdracht JMP≠0202 uitgevoerd. Op deze plek staat normaal het adres van de BREAKHANDLER, te weten ≠C9D0 (kunt u het nog volgen?). Dit adres op ≠0202 kunnen we gemakkelijk veranderen in het adres van onze eigen BREAKHANDLER.
Het idee is nu als volgt: 1)BASIC ontdekt een fout
2)Er wordt een BRK instructie uitgevoerd
3)PC en PS van de 6502 komen op de stack
4)Er wordt gekeken op adres ≠FFFE & ≠FFFF
5)De IRQ/BRK handler springt naar (≠0202)
Het adres op positie ≠0202 en ≠0203 hebben we vantevoren veranderd in het startadres van onze eigen BREAKHANDLER. We vangen als het ware de foutensituatie af om te zien of wij nog wat kunnen maken van de voor BASIC onbegrijpelijke statement. Begrijpen we het (uitbreiding van de BASIC!!) dan voeren we het gewenste commando uit en anders kan er altijd nog naar de standaard BREAKHANDLER teruggesprongen worden.
Hoe weten we nu wat BASIC niet heeft begrepen?
De locaties ≠0005 en ≠0006 geven het begin aan van het laatste te interpreteren statement. Locatie ≠0003 geeft aan hoeveel karakters er vanaf ≠0005 zijn bekeken voordat de BASIC er niets meer van begreep en een BRK instructie genereerde. De nieuwe BREAKHANDLER zal nu vanaf deze positie moeten zoeken (eventueel m.b.v. een tabel) naar een voor hem bekend statement. Wordt het statement herkend, dan moet het uitgevoerd worden. De locaties ≠0005 en ≠0006 moeten daarna verhoogd worden tot op de positie van de eerste punt-komma of 'CR'. Is al dit werk eindelijk gedaan dan kan er teruggesprongen worden naar de BASIC die het werk verder mag opknappen.
Er kan echter niet zomaar naar vast punt in de BASIC teruggesprongen worden, want BASIC kan vanuit verschillende situaties een BRK instructie uitvoeren. Bijvoorbeeld:

```
10 FOR A=1 TO 10;PIEP;NEXT A
20 B=1+3*(4+FAC(5))
30 END
```

In regel 10 wordt een BRK instructie gegenereerd daar PIEP een voor BASIC onbekend 'statement' is, maar in regel 20 wordt een BRK gegenereerd omdat FAC voor BASIC een onbekende 'functie' is.
Bestaat er geen uitbreiding voor het statement 'PIEP' of de functie 'FAC' dan wordt een foutmelding gegenereerd. Een uitbreiding voor 'PIEP' moet na interpretatie terugspringen naar dat gedeelte van BASIC wat een nieuw statement interpreteerd. Een uitbreiding voor 'FAC' mag dat niet doen, daar het huidige statement nog niet beeindigd is. Er moeten nog 2 optel-

lingen gedaan worden, een vermenigvuldiging en een assignment ('B=').

Als start maken we het niet al te moeilijk en zullen we alleen uitbreidingen maken in de vorm van 'statement' uitbreidingen, zoals 'PIEF'. Deze uitbreidingen kunnen allen de BASIC weer inspringen op positie ≠C558
Op deze positie staat

```
C558: JSR ≠C4E4      ;≠5,≠6 ——> eerste ';' of CR
C55B: LDY #≠00       ;kijk of er een ';' staat
C55D: LDA (≠05),Y    ;
C55F: CMP #≠3B       ; =';'
C561: BNE ≠C57D      ;nee, naar nieuwe regel
C563: JMP ≠C31B      ;ja,interpreteer maar weer
```

In routine ≠C4E4 wordt meteen gekeken of de ESC-toets is ingedrukt, en zo ja dan wordt er naar ≠C2CF gesprongen, maar daar hebben wij niets meer mee te maken wat daar gebeurt. De enige voorwaarde die gesteld wordt aan de sprong naar ≠C558 is, dat locatie ≠3 het einde van het zojuist geinterpreteerde statement aangeeft, gerekend vanaf positie (≠0005).
Maar wat nu te doen als de uitbreiding er ook niets zinnigs van weet te maken?

```
Bijvoorbeeld: 10 BLA BLA BLA
              20 END
```

Niets aan de hand, we springen dan gewoon naar positie ≠C9D8, dit is de standaard BREAKHANDLER en die zoekt het verder wel uit.
In de 'PROGRAMMER'S TOOLBOX' van Program Power staat een pracht van een BREAKHANDLER die we met een paar kleine aanpassingen zo over kunnen nemen. Dit is 'm:

```
BREAKHANDLER:
PLP            ;haal de PS en
PLA            ;de lowbyte van de PC
PHA            ;even van de stack en
PHP            ;herstel alles weer,
                maar gebruik de
STA ≠0000      ;PC als error nummer
LDY ≠005E      ;haal het Y-reg weer op
LDA #≠20       ;filter na alle spaties
FILTERSP:
CMP (≠05),Y    ;is het een spatie?
BNE START      ;nee,begin van woord
INY            ;
BNE FILTERSP   ;ja,zoek verder
START:
TYA            ;zet nu ≠5 en ≠6
CLC            ;op het begin van
ADC ≠0005      ;het te interpreteren
STA ≠0005      ;woord
LDA ≠0006      ;
ADC #≠00       ;
STA ≠0006      ;
LDY #≠00       ;initialiseer
STY INDEX      ;adres tabel index
LDX #LENGTE    ;van woorden tabel
ZOEKTAB:
LDA (≠05),Y    ;welke letter is het?
CMP TABEL,X    ;staat hij in de tabel?
BEQ OK2        ;ja,spring weg
CLC            ;truukje,wordt straks
ADC #≠7F       ;uitgelegd!
CMP TABEL,X    ;toch gevonden?
BEQ OK2        ;ja, spring weg
LDA TABEL,X    ;zijn we al aan het
CMP #≠7F       ;eind van het woord
BEQ EIND2      ;ja,gevonden!
BCC OK1        ;nee,doorzoeken
LDA (≠05),Y    ;is het woord soms
CMP #≠2E       ;afgekort?"."
BEQ EIND1      ;ja,en 't is herkend
OK1:
LDA TABEL,X    ;zijn we aan het eind
CMP #≠7F       ;van het woord?
BEQ VERDER     ;ja,volgend proberen
DEX            ;nog niet afgelopen
BNE OK1        ;verder zoeken
VERDER:
DEX            ;zijn we al aan het
LDA TABEL,X    ;eind van de
CMP #≠FF       ;woord tabel?
BEQ FOUT       ;ja,en niet gevonden
INC INDEX      ;volgende poging
LDY #≠00       ;opnieuw woord bekijken
BEQ ZOEKTAB    ;in de tabel
OK2:
INY            ;volgende karakter
DEX            ;bekijken
BNE ZOEKTAB    ;in de tekst en tabel
EIND1:
INY            ;woord gevonden
EIND2:
LDX INDEX      ;haal nr. stat. op
LDA ADRTBL,X   ;lowbyte van adres
```

```
STA ≠0052     ;en highbyte ophalen
LDA ADRTBH,X  ;en wegbergen als
STA ≠0053     ;sprongvector
STY ≠0053     ;Y-reg veilig stellen
JSR INTERP    ;voer statement uit
LDY ≠0003     ;haal Y-reg weer op
DEY           ;
FILTSP:
INY           ;positie verder
LDA (≠05),Y   ;en zoek naar
CMP @≠20      ;niet spatie
BEQ FILTSP    ;verder zoeken
CMP @≠0D      ;einde regel?
BEQ KLAAR     ;ja,klaar zijn we
CMP @≠3B      ;is het een ";"
BNE FOUT      ;nee,dat is fout
KLAAR:
LDX @≠FF      ;maak stack schoon
TXS           ;anders loopt hij over
JMP ≠C558     ;terug naar BASIC
INTERP:
JMP (≠0052)   ;voer statement uit
FOUT:
JMP ≠C9D8     ;naar fouthandler
```

De tabel waarin de nieuwe woorden staan is als volgt opgebouwd (van achteren naar voren!!!):
1) Het betreffende keyword
2) Een byte met inhoud ≠7F
3) Aan het eind v/d tabel een byte ≠FF

Een keyword in de tabel is opgebouwd uit 2 gedeelten, een significant gedeelte, en een niet significant gedeelte. Dit is gedaan om eventueel keywords te kunnen afkorten. Willen we bv. het keyword "PIEP" af kunnen korten tot min. "PI.", dan staat in het significante gedeelte van het keyword de ASCII code voor "PI" en in het niet significante gedeelte de ASCII code plus ≠7F van "EP". Het ziet er dus als volgt uit:

```
 P   I   E   P
≠50 ≠49 ≠C4 ≠CF
```

Als het keyword "PIEP" het eerste element zou zijn, dan moeten de eerste elementen van ADRTBH en ADRTBL het HIGHBYTE en het LOWBYTE van de routine bevatten die "PIEP" uitvoert. Deze routine dient netjes met de opdracht "RTS" terug te keren naar de BREAKHANDLER (zie listing).

De boven beschreven methode om de BASIC interpretator uit te breiden is niet geheel 'foolproof'. Beschouw bv. het volgende onzin programma:

```
10 PIEPERTJE=123
20 END
```

De uitbreiding herkent het keyword "PIEP" en voert dit trouw uit, maar komt dan tot de conclusie dat "ERTJE" niet iets zinnigs voorstelt en geeft weer een piep, maar nu een met "ERROR zoveel" erbij. Maar ja, wie bedenkt er nu zulke kolder?

In het begin is even terloops gezegd, dat de vector op ≠0202 en ≠0203 veranderd wordt door ons om de uitbreiding er in te kunnen linken. Dit heeft echter nog wel wat voeten in de aarde. Probeer maar eens:

```
!≠0202=≠FFFF (ret)
P.&!≠0202
```

Mooi niet gelukt?!?! Dit rare verschijnsel is te verklaren, doordat de BASIC na het inlezen van een (CR) (@≠0D) terugspringt naar het begin van de interpretator en daar de BREAKVECTOR weer terugzet op ≠C9D8. Dit is een schoonheidsfoutje van de Atom vind ik. Dus zullen we, willen we de BREAKVECTOR veranderen, de OSCHRD routine moeten veranderen. Het adres hiervan staat in ≠020A en ≠020B (hier blijft de Atom wel met z'n vingers vanaf). De nieuwe routine moet nu dus steeds als er in "direct mode" een (CR) gelezen is, de BREAKVECTOR veranderen en ietsje verder dan het begin van de interpretator terugspringen. Het constateren of er in "direct mode" gelezen wordt gebeurt m.b.v. de stack. Als n.l. in "direct mode" gelezen wordt, gebeurt dit via ≠CD0F (de line editor) en ≠FFE6 (OSECHO). De line editor wordt opgeroepen in het begin van de BASIC interpretator op adres C2XX (het lowbyte is niet van belang). Als nu 6 posities verder op de stack de byte "C2" staat moet er door de nieuwe OSCHRD de nieuwe

BREAKVECTOR geinstaleerd worden, als er een (CR) ingelezen is. (Sorry als het wat onduidelijk is.) Dit is de nieuwe OSCHRD:

```
OSRDCHR:   JSR ≠FE94      ;lees een character
           CMP @≠0D       ;is er een (CR) gelezen?
           BNE KLAAR      ;nee, niks aan de hand
           TSX            ;haal de stackpointer op
           LDA ≠0106,X    ;kijk of opgeroepen door BASIC
           CMP @≠C2       ;
           BEQ INSTALL    ;zo ja, spring weg
           LDA @≠0D       ;zet accu weer goed
KLAAR:     RTS            ;en terug naar caller
INSTALL:   LDA @≠0D       ;zet accu weer goed
           STA ≠0100,Y    ;(CR) in invoerbuffer
           LDX @≠00       ;
           STX ≠0007      ;zet count op nul
           STX ≠0001      ;idem met regelnummer
           STX ≠0002      ;
           STX ≠0005      ;zet interpretator op de
           INX            ;invoerbuffer
           STX ≠0006      ;
           LDA BRKAL      ;verzet BREAKVECTOR
           STA ≠0202      ;lowbyte
           LDA BRKAH      ;idem met het
           STA ≠0203      ;highbyte
           JSR ≠FEFD      ;naar de nieuwe regel
           JMP ≠C2EA      ;en naar begin van BASIC
```

Nu dan eindelijk de uitbreiding "PIEP":

```
PIEP:      LDX @≠90       ;init. loop
           STX ≠00E5      ;set delay loop
           LDA ≠B002      ;speaker byte
LOOP:      LDY ≠00E5      ;start delay
DELAY:     DEY            ;
           BNE DELAY      ;delay
           EOR @≠04       ;toggle speaker
           STA ≠B002      ;
           DEC ≠00E5      ;verminder delay
           DEX            ;al klaar?
           BNE LOOP       ;nee
           RTS            ;en klaar
```

Nog enige opmerkingen:
Met het boven beschreven "frame" zijn al een groot aantal uitbreidingen mogelijk. Denk bv. aan "PRON" en "PROFF" om de printer aan en uit te zetten etc. Een nadeel is echter, dat er nog geen argumenten overdracht is, bv. "PIEP 3" om drie piepjes te laten klinken. Geen probleem.
Als een uitbreidings routine een getal verlangt, dan kan er naar de subroutine ≠C78B gesprongen worden, waar na terugkeer in de locaties ≠16,X - ≠25,X - ≠34,X en ≠43,X het vier byte integer resultaat staat. De locaties ≠03, ≠05 en ≠06 (zie terug) moeten dan wel naar de goede plek wijzen.
Voor de TOOLBOX mensen kan de uitbreiding beter naar ≠A1E6 terugspringen als er een fout ontdekt is, i.p.v. ≠C9D8, anders worden alle TOOLBOX statements als een fout gedetecteerd!
Het programma kent al het nieuwe statement "PIEP".

Jos Horsmeier.

De JUMPER:

```
>P.&?18
     29> ZING
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
>P.&?18
     29>
```

Dit is geen ingetypte vervalsing: Het is exact wat mijn (G.Borghaerts) Atom momenteel doet als je intypt wat in het v.b. na de prompt ingetypt werd:
a) de vraag: in welke tekstspace zitten we
b) een opdracht
c) dezelfde vraag als a
Het gaat hier óók niet om een "statement" zoals in vorige blz. behandeld door J.Horsmeier, met als voorbeeld "PIEP". Nee, beste clubgenoten, het gaat om DE JUMPER van F.BOUMA. Een betere naam heb ik zogauw niet kunnen bedenken. Waarom die naam?
Wel; het voorbeeldje heb ik met opzet misleidend gemaakt. Het lijkt alsof het hele gezang zich afspeelt in de tekst-space ≠2900 maar dat is nu nét niet het geval. Er wordt als een gek in de ATOM rondgesprongen. Het gezang staat helemaal niet in het 'lage' geheugen,maar in het 'hoge' geheugen (kan ook anders).Kijk maar:

```
>?18=≠82
>
>L.
    5 F.I=1 TO 5
   10 P." H O L A D I J E E !"
   15 P.'
   20 N.
   25 ?18=≠29
   40 END
>
>P.&?18
     82>
>RUN
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
 H O L A D I J E E !
>
> P.&?18
     29>
```

Het gezang staat, in dit v.b., dus in de ruimte ≠82xx. En u ziet dus, hoe na afloop van het gezang in basic wordt terug gesprongen naar de ruimte ≠29xx.
Daarmee is een deel van de truc opgelost.
Blijft de vraag: Hoe kwamen we in ≠82?
U vermoedt het al: met de jumper van Bouma.
Het 'hoe en wat' vereist enige toelichting, ik zal mijn best doen.
Wanneer de Atom een opdracht krijgt, gaat hij op zoek waar hij die opdracht vinden kan en, zo ja, wat ermee gedaan moet. Vindt hij die opdracht nergens, dan komt er uiteindelijk een ERROR 94.
Kort geformuleerd wordt eerst in de "standaard" ROM's gezocht of daar die opdracht te vinden is. Zo niet, bv. bij de opdracht 'FLOAD', dan wordt in de Floating Point gekeken. FLOAD staat daar wel in en die opdracht is uitvoerbaar.
Als de opdracht óók niet in de F.P. wordt gevonden, wordt verder gekeken of er in socket van IC24 een EPROM zit met de inhoud ≠40 en ≠BF op de plaatsen ≠A000 en ≠A001. In dit geval, dan wordt er verder in ≠AXXX gezocht. Was de opdracht 'EDIT' en zit in IC24 de Word Pack EPROM, met daarin het woord EDIT (en wat verder te doen) dan zijn we weer klaar. Zo niet, bv. met de opdracht "PIEP" dan gaat de Atom op weg naar de ERROR-HANDLER. Dáár vangt Jos de zaak op en maakt er alsnog een voor de Atom uitvoerbare opdracht van, een STATEMENT welke als zodanig precies als elk 'gewoon' statement zoals PRINT,RUN,LIST,etc. te gebruiken is en in elk programma. Met de opdracht 'EDIT' ligt dit anders. Gebruik je de opdracht EDIT, dan wordt naar de EDITOR (Word Pack) geschakeld en niet automatisch weer terug. Je blijft dan binnen het EDITOR programma tot je er zelf weer uit wilt, bv. met BREAK.
De JUMPER van BOUMA doet in principe hetzelfde. F.Bouma bedacht een truc om in een EPROM in IC24, dus in geheugen A000 een zelf bedachte opdracht te plaatsen die afgevangen kan worden door de Atom en waarop dan de na de

opdracht volgende instructie wordt uitgevoerd.
Wat het programma JUMPER precies doet, is dat bij elke zelfbedachte instructie, die in EPROM gezet is binnen ≠A000, de TEKST-POINTER wordt gegrepen en gestuurd wordt naar de geheugenruimte, waar het bij deze instructie behorende programma staat en dit meteen RUNt!
Geheugenruimtes worden voor de TEKST-POINTER alleen gedefinieerd door het hogere byte; dus met ≠29, met ≠2A, met ≠82 etc. De beperking van de JUMPER is derhalve, dat via deze jumper alleen Basic-programma's kunnen worden geinitieerd, die op een geheel blok van ≠100 beginnen en vandaaraf gerund kunnen worden. Zie ook nog eens het voorbeeld van het Gezang. Dit had dus óók op bv. ≠37 kunnen staan, als de jumper in ≠A000 de tekstpointer naar ≠37 verwezen had.
Als u het tot zover heeft kunnen volgen, zal het duidelijk zijn, dat de jumper minder geschikt is om als statement te gebruiken. Doch daarentegen uitnemend geschikt om met bv. de opdracht "SORT" ergens in een sorteerprogramma te duiken en deze te doen runnen. En aangezien de 4K op ≠A000 tientallen 'opdrachten' kan bevatten en er vele geheugenplaatsen op 00 eindigen (4 per K) kunnen we met de jumper van Bouma aardig wat doen.

Hoe hanteer je nu dit program. Zelf heb ik, voor het gemak, naar IC24, dus ≠AXXX, een C-MOS IC geadresseerd. Na enig bijschaven kon met de jumper derhalve rechtstreeks een opdracht-codewoord in ≠AXXX gezet worden met daarin de verwijzing (in het voorbeeld naar ≠82). Het bijgaande programma echter gaat in eerste instantie echter naar de ruimte ≠3XXX (zie regel 50 en 100 van het programma). D.w.z. het gaat met *LOAD naar ≠3100 dan intypen END en dan RUN. Het assembleert aan zichzelf vanaf ≠3002 tot en met ≠3041, gevolgd door het gekozen codewoord. Bij mij dus ZING. Op ≠3000 zet het programma ≠40 en op ≠3001 komt ≠BF. Op ≠3042 staat het door u gekozen verwijsadres voor de TEKST-POINTER. Het program vraagt u de gegevens en laat het resultaat keurig zien.
Kortom: Vanaf ≠3000 staat dus geheel precies alles, wat daarna in EPROM vanaf ≠A000 moet staan. Het program geeft zelfs de adressen, die u in de conversatie met de ZERO-PROGRAMMER dient in te vullen.
Noteert u ergens precies de opdrachten en adressen!!
Wilt u later een tweede of zoveelste CODEWOORD toevoegen, dan is het 't gemakkelijkst om met het nieuwe codewoord achteraan toegevoegd, alle codewoorden opnieuw in EPROM te programmeren. De programmer overschrijft dan precies wat er reeds staat (kan geen kwaad) en voegt aansluitend het nieuwe codewoord toe. U hoeft niet eerst te wissen! Houd dus dezelfde woorden in dezelfde volgorde aan!!

DE JUMPER 2. ******

Natuurlijk kon het voorgaande niet het eindresultaat van de JUMPER blijven, door Foppe Bouma werd hard gesleuteld, met het volgende resultaat: De Nieuwe JUMPER een herziening en uitbreiding van de vorige. Bekijken we de mogelijkheden van dit program (= EPC.FB) eens. Het program heeft 7 functies:
1) jumper naar basic-programma's. Deze functie is identiek aan wat de vorige JUMPER deed. Nu moeten echter zowel low-byte als high-byte van het start-adres worden opgegeven. De low-byte moet altijd 0 zijn!! Daaraan herkent het programma, dat het om de uitvoering van een BASIC-programma gaat. Dat soort basic-programma's kan het beste worden afgesloten niet met END, maar met het nieuwe statement HOME, wat ik heb gemaakt (zie file MONITCOM.FB). Met dit statement springt de textspacepointer terug naar ≠29 en wordt eveneens TOP weer correct gezet.
2) jumper naar machinetaal-programma's. Ook van dit start-adres moeten zowel low- als high-byte worden opgegeven. Hier (en ook voor de volgende statements) geldt echter, dat de low-byte alle waarden tussen 1 en 255

mag hebben, maar niet 0!!! Het machinetaal-program moet eindigen met een jump naar de monitor via JMP≠C2CF.
3) uitvoering van monitor-commands in de direct-mode. Dit is in feite identiek aan punt 2. Zie bv. de commands HIGH en VHY, die een tekstspace creeren op resp. ≠8200 en ≠9800, daar het commando NEW uitvoeren, indien nog geen tekst aanwezig is en de variabele TOP goed zetten. Ook de commands TS en TP, die resp. laten zien waar de tekstspacepointer staat en waar TOP (zie file MONITCOM.FB)
4) uitvoering van extra statements in basic programma's zonder argumenten. Zie bv. PON, POFF, SON en SOFF, die resp. de printer of de screen aan cq. uit zetten. Als je zo'n statement maakt, moet je aan het eind van de routine terug springen naar ≠C558.
5) uitvoering van extra basic-statements met argumenten. Op mijn systeem werkt nu bv. al het statement FMT[$A,8.2]. Dit format-statement heeft dezelfde functie als in FORTRAN. Het transformeert een real getal in $A (bv. gevormd via STR%X,A) in een afgerond getal met een field width van 8 en 2 cijfers achter de komma.
6) start automatisch de toolbox via het eerst gegeven toolbox commando, of via de eerste return na BREAK (dit is belangrijk bij de cassette-load routines!), waarbij een willekeurig ERROR-nummer wordt vertoond.
7) een ingebouwd DIR-statement die een listing geeft van de ingebouwde commando's in de betreffende EPROM, inclusief de start-adressen ervan. Dit is vooral handig bij meerdere gemultiplexte EPROMs op het zelfde adres (zie als vb. de DIR-listing).
8) een extra facultatieve doorverbinding in het zoekproces van commando's naar het RAM-geheugen of in de toekomst naar het EPROM-deel van bv. ≠400 tot ≠2800. (D.i. de file ECRAM.FB)

De file EXTC.FB is bedoeld om uitbreiding van nieuwe statements aan een al gedeeltelijk gevulde EPROM op gemakkelijke wijze toe te voegen.

Foppe Bouma.

file ECRAM.FB

```
 10P.$12"BASICCODES V EPROMS"';G.40
 20F.BOUMA BEELLANEN 71
 40 DIM LL9,B15
 45 S=≠3A00;REM SOURCE ADR
 46 D=≠3A00;REM EPROM BASE
 52 Z=≠62;W=D+Z;V=S+Z
 55P.$21;G.60
 59GOS.z;P.$6;GOS.z;E.
 60 GOS.z;GOS.z;P.$6;G.a
100zP=S+2
110[
115 LDX@0
120:LL0 LDA W,X;STA≠90;INX
125 LDA W,X;STA≠91;INX
130 LDY≠03;DEY
140:LL1 INY;LDA(≠05),Y
150 CMP@≠20;BEQ LL1
160:LL2 LDA W,X;BMI LL4
170 CMP (≠05),Y;BNE LL3
180 INX;INY;BNE LL2
190:LL3 INX;LDA W,X
200 CMP@≠FF;BNE LL3
210 INX;LDA W,X
220 CMP@≠FF;BNE LL0
230 LDA≠020A;CMP ≠94
232 BNE LL7;JSR≠AF00;JMP≠A1E6
235:LL7 JMP≠C558
240:LL4 LDA(≠05),Y;CMP@≠5B;BEQ LL8
245 STY≠03;PHA;JSR≠C4E4;PLA
250:LL8 LDA≠90;BNE LL6
260 LDA≠91;STA≠12;JMP≠CE86
270:LL6 JMP (≠0090)
275];R.
300aIN."AANTAL CODEWOORDEN"N
310I=0;F.L=1TON
320P."CODEWOORD "L;IN.$B
330IN."LOW BYTE "T,"HIGH BYTE"H
340V?0=T;V?1=H;V=V+2;$V=$B
350V?(L.B)=≠FF;V=V+L.B+1;N.
360V=V-1;M=V-W;F.I=0TO M;K=W+I
370P.&K"  "&?K;IF?K>31P."  "$?K
380P.';N.
390?(V+1)=≠FF
399E.
400P.''"ADRESSEN V ZERO-PROGRAMMER"'
    "FIRST MEM=≠"&S'
410P."LAST  MEM=≠"&V'"FIRST PROM=
       0"';E.
```

file EPC.FB

```
 10P.$12"BASICCODES V EPROMS"';G.40
 20F.BOUMA BEELLANEN 71
 30 3445 TE WOERDEN:26-6-'82
 40 DIM LL18,B15
 50 S=≠3000;REM SOURCE ADR
 60 D=≠E000;REM EPROM BASE
 70 F=D-S
 80 ?S=≠FB;?(S+1)=≠BF
 90 Z=≠8F;W=D+Z;V=S+Z+6;U=V
100 !(S+Z)=≠4944E063
110 !(S+Z+4)=≠FFFFFF52
120P.$21
130 GOS.z;GOS.z;P.$6;G.a
140zP=S+2
150[
160 LDX@0
170:LL0 JSR LL16+F
180 LDY≠03;DEY
190:LL1 INY;LDA(≠05),Y
200 CMP@≠20;BEQ LL1
210:LL2 LDA W,X;BMI LL4
220 CMP (≠05),Y;BNE LL3
230 INX;INY;BNE LL2
240:LL3 INX;LDA W,X
250 CMP@≠FF;BNE LL3
260 INX;LDA W,X
270 CMP@≠FF;BNE LL0
280:LL8 LDA≠0211;CMP@≠C2
290 BEQ LL5;JMP (≠0210)
300: LL5 LDA≠020A;CMP@≠94
310 BNE LL7;JSR≠AF00;JMP≠A1E6
320:LL7 JMP≠C55B
330:LL4 LDA(≠05),Y;CMP@≠5B;
     BEQ LL13
340 STY≠03;PHA;JSR≠C4E4;PLA
350:LL13 LDA≠90;BNE LL6
360 LDA≠91;STA≠12;JMP≠CE86
370:LL6 JMP(≠0090)
380 LDX@6;STX≠92
390:LL11 LDY@0;JSR LL16+F
400:LL9 LDA W,X;BMI LL15
410 JSR≠FFF4;INX;INY;BNE LL9
420:LL15 JSR LL10+F;LDA≠91;
     JSR≠F802
430 LDA≠90;JSR≠F802
440 INC≠92;LDA≠92;AND@1
450 BEQ LL17;LDY@2
460 JSR LL10+F;BCS LL12
470:LL17 JSR≠FFED
480:LL12 INX;LDA W,X;CMP@≠FF
490 BNE LL11;JSR≠FFED;JMP≠C2CF
500:LL16 LDA W,X;STA≠90;INX
510 LDA W,X;STA≠91;INX;RTS
520:LL10 CPY@8;BCS LL14
530 LDA@32;JSR≠FFF4;INY;BNE LL10
540:LL14 RTS
```

vervolg EPC.FB

```
550];R.
560aIN."AANTAL CODEWOORDEN"N
570V=0;F.I=1TON
580P."CODEWOORD "I;IN.$B
590IN."LOW BYTE "T,"HIGH BYTE"H
600V?0=T;V?1=H;V=V+2;$V=$B
610V?(L.B)=≠FF;V=V+L.B+1;N.
620V=V-1;M=V-U;F.I=0TO M;K=U+1
630P.&I"  "&?K;IF?K>31P."  "$?K
640P.';N.
650?(V+1)=≠FF
660P.''"ADRESSEN V ZERO-PROGRAMMER"'
   "FIRST MEM=≠"&S'
670P."LAST  MEM=≠"&V'"FIRST PROM=
      0"';E.
```

file MONITCOM.FB

```
 10REM MONITOR COMMANDO'S
 12 ?=0
 15DIM LL10
 20GOS.z;GOS.z
 25F.I=0TO9;P."LL"I,"  ≠"&LLI';N.
 29E.
 30zP=≠324D
 40[
 50:LL0 LDA@≠29\HOME
 60 STA≠12;BNE LL4
 70:LL1 LDA@≠82\HIGH
 80 BNE LL3
 90:LL2 LDA@≠98\VHY
100:LL3 STA≠12;STA≠91
110 LDY@0;STY≠90
120 LDA (≠90),Y;CMP@13;BEQ LL4
125 JSR≠F7D1
130];$P="NEW";P=P+L.P;[
135 NOP;JMP≠C2B6
140:LL4 JMP≠CD9B
150:LL5 JSR≠F7D1\TP
160];$P="TOP:≠";P=P+L.P;[
170 NOP;LDA≠0E;JSR≠F802
180 LDA≠0D
190:LL7 JSR≠F802;JSR≠FFED
195 JMP≠C2CF
200:LL6 LDA@≠23;JSR≠FFF4\TS
210 LDA≠12;BNE LL7
220]
230R.
```

```
file EXTC.FB
 10P.$12"BASICCODES V EPROMS"';G.40
 20F.BOUMA BEELLANEN 71
 40 DIM LL15,B15
 45P."BEGIN BASE:≠3000"';S=≠3000
 47 IN."FREE BASE:"V
 50 U=V
300aIN."AANTAL CODEWOORDEN"N
310@=0;F.I=1TON
320P."CODEWOORD "I;IN.$B
330IN."LOW BYTE "T,"HIGH BYTE"H
340V?0=T;V?1=H;V=V+2;$V=$B
350V?(L.B)=≠FF;V=V+L.B+1;N.
360V=V-1;M=V-U;F.I=0TO M;K=U+I
370P.&K"  "&?K;IF?K>31P."  "$?K
380P.';N.
390?(V+1)=≠FF
400P.''"ADRESSEN V ZERO-PROGRAMMER"'"FIRST MEM=≠"&U'
410P."LAST  MEM=≠"&V'"FIRST PROM=",&(U-S)';E.
```

```
file BAFK.FB
 10REM BEGIN A000-EPROM
 20DIM LL2
 30S=≠3000;REM SOURCE ADR
 40D=≠E000;REM DESTINATION
 50?S=≠40;?(S+1)=≠BF
 60GOS.z;GOS.z;E.
 70zP=S+2
 80[
 90LDA D;CMP ≠FB;BNE LL1
100LDA D+1;CMP ≠BF;BNE LL1
110JMP D+2
120:LL1 JMP≠C558
130];R.
```

voorbeeld:

DIR

BLOK	E200	HOME	E24D
HIGH	E253	VHY	E257
TP	E274	TS	E28D
TRY	3A01	HRUN	8200
VRUN	9800	PON	E296
POFF	E29A	SON	E29E
SOFF	E2A2	RAMON	E2AA

ARCHIEFBANDJE nr.3 een VERZAMELING DISASSEMBLERS

****

"DA.AM" = Disassembler A.Marchal. Een disassembler in Basic, te laden naar ≠2900 of *LOAD≠8200, doet er niet toe. Het program beslaat iets minder dan 3K. Geeft men na het gevraagde beginadres het gewenste aantal bytes negatief (met -) op, dan is het een orthodoxe disassembler. Geeft men het aantal gewwon (pos.) op, dan gaat het program, wanneer iets niet begrepen wordt over op byte voor byte en vertaalt naar alphanummeriek.
"DA.AC" = Disassembler van de Acorn-fabriek. In de verzameling opgenomen uitsluitend ter bestudering. Het program is kort, precies 1k. Voor haastige mensen. Het program moet naar ≠82 en laadt met *RUN. Er zit een BRK option in en een directe copieermogelijkheid naar ≠29 of elders.
"DAMO(OBJ).RH" = Een disassembler-Monitor van R.Heuvel. Een juweel! Laat vele instructies toe als S(tep),T(race),L(ist) etc. Geeft de inhoud weer van alle REGISTERS en de FLAGS. Volgt naar wens de uitvoering van de instructies. Komt hij dus ooit de instructie tegen 'clear screen', (dit als v.b.) dan ziet u gelijk niets meer. Op het bandje staat het program in object-code maar ook als:
"DAMO(SRC).RH)" = in Source. Dit voor onderzoekers en 'doe het zelvers'. In objectcode met *LOAD naar ≠37F0 tot ≠3C00, dus precies 1K. In source uiteraard meer; van ≠8200 tot ≠9513 is 3½K. Met het source program naar wens te veranderen.
"DA.MI" = Disassembler van A.Millenaar. Een 'recht toe recht aan' disassembler in Basic, gewoon naar ≠29 met LOAD. Iets meer dan 3K. Het program is probleemloos te listen en te printen en voor iedereen te begrijpen. Een prima kennismaking met de disassembler-methodiek.
"DA B.JH" = Disassembler, Basic gedeelte. Van J.Horsmeier. Gaat naar ≠2900 tot ≠2CDA, dus iets minder dan 1K. Het program gebruikt tabellen "DA T.JH" = via precies 255 bytes aan DATA. Voor het gemak volgt op *LOAD"DA T.JH" 2800 (op het bandje) het Basic deel er automatisch achteraan, zodat u met 1x laden klaar bent. Totaal 1 1/4K. Na laden BREAK (ret) OLD (ret) L=≠AAAA,beginadres (ret) H=≠EEEE,eindadres (ret) T=≠TTTT, tabeladres, nu dus ≠2800. Het program zet het scherm in page-mode. Voor u gemakkelijk. Is eruit te halen: lijnnr.0.

ARCHIEFBANDJE nr. 9. *********

MONITOR van Roel Heuvel.
Op dit bandje een intussen verbeterde versie. Oorspronkelijk gaat het hier om de Apple Monitor. Roel heeft deze geschikt gemaakt, practisch omgebouwd, voor gebruik in de Acorn Atom. Deze monitor is een stuk gereedschap om in geheugens te kijken en in de registers. Deze monitor wordt gdruikt door Bijnagte in de cursus Assembler in Acorn Nieuws.
Van deze monitor staat op het bandje op de eerste plaats de Source. Source betekent bron of oorsprong. Deze source is op zich een programma. Het wordt geladen met *LOAD"RH SRC" 8200 geladen naar ≠8200 en vult dit hoge geheugen tot ≠9561. Bij RUN van deze Source assembleert dit programma de eigenlijke monitor die slechts 1K lang is naar een opgegeven plaats, de plaats, waar men dit gereedschap in zijn computer hebben wil.
Het beginadres van deze 1K monitor staat in de source in regel 150. Daar staat P=Z+≠C00 P is dit beginadres. Volgens deze opdracht is P dus C00 d.i. 3K Hoger dan het adres Z. Het adres Z staat in regel 135. Daar staat Z=≠2C00. Het beginadres P wordt in dit geval dus ≠2C00+≠C00=≠3800. De feitelijke monitor wordt door de source in dit geval dus neergezet vanaf het adres ≠3800 en eindigt 1K later, d.i. dus op adres ≠3BFF. Dat is een goede plaats voor mensen, die het lage geheugen (nog) niet gestapeld hebben.
Is dit lage geheugen wel uitgebreid van ≠3C00 tot ≠3FFF, dan kan men de

monitor beter hoger zetten, vanaf ≠3000. Dit kan eenvoudig bereikt worden door in de source regel 135 te veranderen in: Z=≠3000;REL BASE ADDRESS Heeft men de 16K CMOS-geheugenkaart, dan zou de monitor op ≠3000 weer in de weg staan. Een betere plaats is dan ≠7000. Te bereiken door in de source in te vullen: Z=≠7000;etc. Heeft men de schakelkaart dan kan de monitor nog beter verhuizen naar ≠A000 in EPROM of CMOS of zo. De source die na RUN de eigenlijke monitor heeft samengesteld en weggezet, kan dan verdwijnen. De eigenlijke monitor kan men dan op de band zetten met (bv.) *SAVE"MRH MC"3800 3BFF 3AD5 Het adres 3AD5 is het executieadres en ligt bij de monitor altijd ≠2D5 locaties hoger dan Z. De oorspronkelijke source versie was bedoeld om de monitor direct weg te zetten in EPROM. Hiertoe stond in de source de regel 110, die de gehele 4K EPROM eerst wist voor de zekerheid. Bij schrijven naar gewoon geheugen kan men die regel beter eerst verwijderen. Ook regel 155 diende dit doel. Die regel kan men gewoon laten staan en op het commando 'Schakelaar omzetten' returntoets geven.
De source maakt dus de monitor en zet deze ergens weg. Hij 'run't de monitor ook meteen, zodat na het geven van de assemblerlisting de dubbele punt (:) verschijnt.

SCHIPHOL JW.

Het betreft hier een FLIGHTSIMULATOR, een door Jaap van de Woestijne omgeschreven versie van het oorspronkelijke program "747" van Bug Byte.
De bedoeling van het program is, dat men als piloot zittend in de cockpit van de PH-BUX volledig op instrumenten vliegt naar één van de peilstations rond Schiphol en zorgt daar aan te komen op de goede hoogte en met de goede snelheid etc. Dan aldaar op kompas de bocht draait naar de landingsbaan. Vliegt men dan inderdaad precies door het voorgeschreven 'raam' dan kan de landingsbaan in de verte worden gezien. Bij de goede daalsnelheid komt men netjes terecht en wordt de prestatie in procenten en een compliment uitgedrukt. Gaat het echter te hard of te langzaam met dalen, dan kwakt men tegen de grond of vliegt erover heen. In beide gevallen zet het programma uw toestel echter weer netjes op de baan en wordt u verzocht om weer te vertrekken. Het programma is veel duidelijker en verloopt vele malen soepeler dan het oorspronkelijke program van Bug Byte. Het is ook gemoderniseerd in taalgebruik en aangepast aan de Schiphol procedures. Het vliegen op 2 peilstations tegelijk, één 'AHEAD' en één voor kruispeiling vergemakkelijkt de orientatie op instrumenten ook aanzienlijk.
Het programma is lang en vraagt minsten het 'gestapelde' lage geheugen vanaf ≠2900 dus. Het programma is door Jaap in opzet gelocaliseerd naar ≠4000, het beginadres van de CMOS-geheugenkaart en loopt vandaar prima, het beeld is 'smetteloos'. Het vereist ook de Floating Point, zodat men niet vanaf ≠2800 kan laden. Door de waanzinnige hoop berekeningen kan er een enkele maal in het program wat mis gaan. Dan gewoon opnieuw runnen. Jaap leverde bij het program de benodigde 'LANDING APPROACH' kaarten bij van Schiphol. Foto-copieen van deze kaarten kan men bestellen bij het 'listings-archief' f2,- in postzegels, of in de regio.
Vluchtsimulatoren zoals deze (maar dan wat groter dan een Atom) worden bij luchtvaartmaatschappijen gebruikt voor trainen van piloten. Bij deze versie van J.van de Woestijne gaat het zeker niet om een spelletje, maar zult u letterlijk de handen vol hebben.
Gebruik van de toetsen:
Toets '1' en toets '2' kiezen de peilstations. Advies: kies met '1' het station voor een van de 4 ADF-bakens OA, WF, CH of SNV. De koers die u vliegen moet om dit baken te bereiken staat aangegeven bij '1' en is in te stellen door de koers onder 'HDG' (heading) daaraan gelijk te maken. Dit doet u door een bocht te draaien; toets LOCK linksom; REPT rechtsom.

Visueel blijft uw toestel maar draait de horizon weg en zo hoort het. Of de koers nu correct is ziet u op het buitenste knipperlichtje van het compas. Het binnenste (dichtst bij centrum) geeft de richting aan van peilstation '2'. Als het goed is, vermindert nu de afstand tot het ingestelde baken. Uiteindelijk tot 0. U zit er dan recht boven.
U moet nu een aantal dingen tegelijk doen. Een paar maar. De hoogte dient op dat moment ca. 2000 Voet te zijn. De landingsklappen moeten uit met '+' (vasthouden). De wielen moeten uit met 'G'(gear). Door deze handelingen loopt de snelheid terug. Kom met klappen 'in' niet onder 220 knopen en met klappen vol uit (30 gr.) niet onder 140 kn. Z.n. gas meerderen met ↑ of minderen met 'L'(less). Vanaf het baken dient u exact op koers te gaan naar de landingsbaan. Die koers ziet u op APPROACH kaart nr.6. Vanaf baken OA is dit dus (bv.) 187 graden. (0 is Noord).
Zit alles goed, dan vliegt u nu door een RADAR-raam waardoor de instrumenten verdwijnen. Is de koers exact goed, dan ziet u in de verte de landingsbaan als een stip en geleidelijk groter. Met 'H' neigt de neus naar beneden; met 'spatie' kunt u weer optrekken. Zet vóór de instrumenten verdwijnen met H de ATTITUDE op -3 en corrigeer met H en spatie de daalsnelheid op zicht eventueel bij. Richt daarbij op de hoogte van de horizon, flikkerstreepje links, moet ongeveer halverwege de schermhoogte blijven.
Bij vertrek: - is 'flaps in' (pas als u 220 kn. heeft!)
G is 'wielen in' (pas als u los bent, zie hoogte)

IAS= Indicated Airspeed (snelheid)
ALT= Altitude (hoogte)
FLAP=klappen
ATC= Air Trafic Control (Schiphol)
ROLL=draaien om lengte-as
N1%= Turbine rotatie (gas)
V/S= stijg/daalsnelheid
Rad= Radiaal
HDG= koers
ATTITUDE=draaien om dwarsas (helling)

F.I.FORTH.
Als andere 'hogere programmeertaal' dan Basic is voor de Atom beschikbaar de taal FORTH. De programmeertaal FORTH is beter dan Basic geschikt om daarmede programma's te schrijven t.b.v. bv. schakeldoeleinden, zoals bv. het sturen van een modelspoorbaan. FORTH wordt dan ook nogal eens gebruikt in de Meet en Regeltechniek.
In Amerika bestaat er een FORTH gebruikersvereniging, de FORTH INTEREST GROUP, die een goede gebruikers-FORTH heeft samengesteld en deze vrij ter beschikking stelt. Ons lid ter Harmbel zond mij deze F.I.G.FORTH toe aangepast aan de Atom routines. Volgens Akkermans, die de cursus FORTH verzorgt, is deze zeker zo goed als FORTH van Acorn-soft.
U laadt dit programma met *LOAD"FORTH DTH"2800 Daarna met *LOAD"PAGE 2" 0800 de gegevens naar Block 0 RAM. Dan start u FORTH met koude start door LINK≠2800 en met warme start door LINK≠2804. De cursor komt dan niet terug, doch als u dan intypt DON dan krijgt u een listing van de nestings en volgt de dubbele punt (:) als cursor.
Attentie: Bij laden van de FIG FORTH schrijft het programma vanaf ≠2800 tot aan ≠3FFF. U dient dus wel een gestapeld laag geheugen te hebben. Het programma schrijft ook heen over de geheugenplaatsen ≠2900 en ≠2901. Op ≠2900 staat dan ≠15 en op ≠2901 staat ≠06.
Wanneer u bij gebruik van deze FORTH echter een keer de BREAKtoets gebruikt, dan worden deze 2 plaatsen vernield. Pook dan even de code terug.
DAMMEN 1 en 2 van BOOGAARD.
Dammen 1: Laadt van ≠8200 tot ≠9B20 en vraagt dus gestapeld hoog geheugen.
Dammen 2: Laadt van ≠2C00 tot ≠3D21 daarna ?18=≠35 END RUN
De notatie is van LINKS BOVEN af de Bruine velden met 1 2 3 4 enz. Tot onder rechts 50. (Vul bij de vraag 'meerslag' een cijfer in en niet JA).

ASSEMBLER CURSUS

DEEL 1:
Deze kursus Assembler is speciaal voor hen die de Atombasic behoorlijk beheersen. In de komende afleveringen zullen de Atom assembler en de instructieset behandeld worden, zodat ieder, na het serieus doorwerken van de kursus, zich behoorlijk thuis zal voelen in machinetaal programmeren. Net als voor de basic geldt:" Oefening baart kunst". Van alleen theorie zal men nooit de puntjes op de i, of onder ons gezegd;"de bits in het byte" krijgen. Als gereedschap voor deze kursus is het Archiefbandje 6 met het monitorprogramma van R.Heuvel bijzonder handig. Vooral als we met het behandelen van Flags gaan beginnen.
Het hart van onze Atom: de 6502
De 6502 is de uitvoerder van alle opdrachten. Dit geldt voor zowel basic als voor de assembler. De 6502 doet alleen werk wat hem opgedragen wordt. Wat hij precies moet doen, ligt vast in de opdracht. Het scala van opdrachten is gespecificeerd in een zgn. instructieset. Zo'n instructieset bevat dus alle machinetaalmogelijkheden van de 6502. Als de 6502 dus een kode tegenkomt, dan voert hij dit uit. Het is dus een echt dom ding: wat hij doet is alleen uitvoeren wat van hem verlangd wordt. Enig greintje zelf-initiatief toont hij (gelukkig) niet. U vraagt zich nu natuurlijk af:"Ho effe, en als ik nu intyp Print "Hallo", wat doet 'ie daar dan mee?" Nou, dat is in principe heel eenvoudig. Het commando Print wordt omgezet in een aantal machinetaal-instructies en die worden op hun beurt weer aangeboden aan de 6502 en die voert het uit.
De Atom assembler:
Deze assembler is uw rechterhand bij het maken van machinetaalprogramma's. Wat dit ding doet, is in feite erg simpel. Op de huis-, tuin-, en keukenmanier typt u uw opdrachten (instructies) in, alsof u een basicprogramma intypt. Nadat het programma is ingetypt, wordt het "gerund". Nu komt de eigenlijke assembler in actie. Instructie voor instructie wordt vertaald in hexa-decimaal-getallen en opgeslagen, op een door u te bepalen stukje geheugen. U hoeft dus niet zelf al die vervelende hex-getallen in te typen, dat wordt voor u gedaan.
Hexadecimale getallen:
De 6502 werkt met hexadecimalen. D.w.z. 0-9 ABCDEF. Als op een geheugenlokatie een getal moet worden opgeslagen, is dit een hex-getal. Op één geheugenplekje kan een getal van 8 bits worden opgeslagen. Een geheugenplaatsje kan dus een getal tussen 00 en ≠FF (0-255) bevatten, want het is een 8 bits, dus er zijn 2^8 mogelijkheden, 2x2x2x2x2x2x2x2. Willen we zelf een geheugenplaatsje vullen, dan kunnen we dit doen met het vraagteken. B.v. ?≠90=≠2A. Als we willen controleren of het klopt, toetsen we in PRINT & ?≠90 en voilà: 2A. Een iets meer zichtbare methode is om het beeldscherm als geheugen te gebruiken. Toets eens in ?≠8000 =≠2A. Links bovenin ziet u een sterretje verschijnen. Probeer dat ook eens met andere waardes. Opvragen van dit geheugenplekje is eenvoudig. Toets in P.&≠8000 (zorg wel dat de bovenste regel niet 'van het scherm valt', anders wordt ook de geheugenplaats 'overschreven' en zakt m'n hele theorie in het ijs.)
Een klein stukje theorie van de 6502:
Voordat we met machinetaalprogrammeren beginnen, moeten we eerst de meest essentiële dingen van de 6502 weten.
- 3 (8 bits) interne registers de Accu X en Y. Dit moet je zien als b.v. Z in basic. In deze geheugens kunnen getallen tussen 0 en 255 (00-FF).
- Een vlaggenregister of statusregister. Hierin bevinden zich de standen van de vlaggen. Het volgende deel zal over vlaggen gaan.
- De Program counter. Dit is een register wat weergeeft waar de volgende

instruktie zich bevindt. Het is een 16bits register en kan dus variëren tussen 0000 en ≠FFFF.
-Memory. Dit register wordt door de gebruiker bepaald. Een memory kan in principe alles tussen 0000 en ≠FFFF zijn. Als u dus opdracht geeft een getal op ≠97FF op te slaan, dan is de memory tijdens de instruktie ≠97FF.

Machine instructions.

ADC Add Memory to Accumulator with Carry
AND AND Memory with Accumulator
ASL Shift Left One Bit (Memory or Accumulator)
BCC Branch on Carry Clear
BCS Branch on Carry Set
BEQ Branch on Result Zero
BIT Test Bits in Memory with Accumulator
BMI Branch on Result Minus
BNE Branch on Result Not Zero
BPL Branch on Result Plus
BRK Force Break
BVC Branch on Overflow Clear
BVS Branch on Overflow Set
CLC Clear Carry Flag
CLD Clear Decimal Mode
CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag
CMP Compare Memory and Accumulator
CPX Compare Memory and Index X
CPY Compare Memory and Index Y
DEC Decrement Memory by One
DEX Decrement Index X by One
DEY Decrement Index Y by One
EOR Exclusive-OR Memory with Accumulator
INC Increment Memory by One
INX Increment Index X by One
INY Increment Index Y by One
JMP Jump to New Location
JSR Jump to New Location Saving Return Address
LDA Load Accumulator with Memory
LDX Load Index X with Memory
LDY Load Index Y with Memory
LSR Shift Right One Bit (Memory or Accumulator
NOP No Operation
ORA OR Memory with Accumulator
PHA Push Accumulator on Stack
PHP Push Processor Status on Stack
ROL Rotate One Bit Left (Memory or Accumulator)
ROR Rotate One Bit Right (Memory or Accumulator)
RTI Return from Interrupt
RTS Return from Subroutine
SBC Subtract Memory from Accumulator with Borrow
SEC Set Carry Flag
SED Set Decimal Mode
SEI Set Interrupt Disable Status
STA Store Accumulator in Memory
STX Store Index X in Memory
STY Store Index Y in Memory
TAX Transfer Accumulator to Index X
TAY Transfer Accumulator to Index Y
TSX Transfer Stack Pointer to Index X
TXA Transfer Index X to Accumulator
TXS Transfer Index X to Stack Pointer
TYA Transfer Index Y to Accumulator
PLA Pull Accumulator from Stack
PLP Pull Processor Status from Stack

Compare Instruction Results.

Condition	N	Z	C
A,X, or Y < Memory	1	0	0
A,X, or Y= Memory	0	1	1
A,X, or Y > Memory	0	0	1

DEEL 2:
De Stack pointer.
Dit is een 8-bits register. Dit register vormt samen met een vast getal (≠01) het eerste vrije stack adress. De stack wordt door de processor gebruikt om tijdelijke gegevens op te slaan. B.v. het terugkeer adres van een subroutine. Stack adress 01xx - 01FF
Assembler variable P.
De ruimte waar het programma moet worden geassembleerd moet door de gebruiker worden bepaald. Moet het pal achter het programma komen dan

wordt het DIM P-1. D.w.z. dat het programma pal achter TOP geassembleerd wordt. Weet je van tevoren al een adres, dan geef je die waarde aan P. Dus b.v. P=#2800. Zorg dat bij een DIM-instruktie DIM P-1 de laatste instruktie is, omdat de ruimte(=lengte) niet vast ligt, die voor het programma nodig is. Met P legt u n.l. het beginadres vast.

Haken.

Het programma dat in machinetaal moet worden gezet, moet tussen haken worden gezet. Zo'n haak maakt de interpreter duidelijk, dat er een assembler programma tussen staat. Vergeet u de openingshaak, dan verschijnt (piep) ERROR 94 in line XX. Bij vergeten eindhaak ERROR 208 in line XX. Een programma ziet er dus minimaal zo uit:

```
 10 P=#2800    of DIM P-1
 20[
 --
 --
100]
110 END
```

Vlaggetjes dag.

Het status of vlaggenregister is een belangrijk onderdeel van de 6502. Een vlag is een hulpmiddel voor de programmeur. Een vlag kan 1 of 0 zijn. Als de processor daartoe opdracht krijgt, kijkt hij of de genoemde vlag 0 of 1 is. U kunt hem vergelijken met de IF-THEN. Als de voorwaarde achter IF waar is (1 is dus) THEN ... Na iedere instruktie wordt het statusregister bijgewerkt.

Indeling statusregister.

N/V/-/B/D/I/Z/C/

De vlaggen stuk voor stuk:

De carry vlag C

Deze vlag wordt gezet als bv. bij een optelling de #FF wordt overschreden. U kent hem vast wel van de lagere school: één onthouden als de optelling groter is dan 9. Bij aftrekken is het net andersom. De carry wordt 0 als er 'geleend' wordt; bv. 10014-9=10005. Bij schuiven en roteren kan de carry worden gebruikt als 9e bit.(Hierover volgt 'n aparte aflevering).

De zero vlag Z

Deze vlag wordt gezet (1) als de voorafgaande bewerking op 0 uitkomt. Dus bij 5-5=0 wordt de zero vlag gezet.

Interrupt bit I

Dit is eigenlijk een apart verhaal. De processor bezit een ingang IRQ. Als deze ingang 1 wordt, kan de processor, afhankelijk van de stand van de vlag, gedwongen worden om een ander stukje programma uit te gaan voeren. U kunt de vlag vergelijken met een direkteur: Als de vlag 1 is hangt het bordje 'niet storen' op de deur. Als de vlag 0 is mag er wel gestoord worden. Het storen gebeurt niet direct. De processor maakt eerst zijn instruktie af en kijkt dan naar de interrupt. (t_{max}= 3microsec.)

Decimal/Hex vlag D

Als deze vlag gezet is, zullen op- en aftellingen decimaal gebeuren. Is de vlag 0 dan zullen alle rekenkundige bewerkingen hexadecimaal geschieden.

Break vlag B

Als de processor tijdens een programma een BRK-instruktie tegenkomt dan wordt deze vlag gezet en springt hij via de brk-vector weg. Wordt ook wel software interrupt genoemd.

Bit 5 -

Wordt niet gebruikt.

Overflow vlag V

Deze vlag wordt gezet als een bewerking groter is dan 64 (#3F).

Negative vlag
Deze vlag wordt gezet als het MSB van een getal 1 is; groter dan 128 (≠7F). Als bv. een getal wordt afgetrokken van een getal en de N-vlag wordt hierbij gezet, dan is het getal dus negatief.

Sommige vlaggen kunnen ook door de programmeur worden 'geset' of 'reset':
CLC clear carry (maak de carry vlag 0) SEC set carry (maak de carry vlag 1)
CLD clear decimaal (maak de D-vlag 0) SED set deci. (maak de D-vlag 1)
CLI clear interrupt(maak de I-vlag 0) SEI set inter.(maak de I-vlag 1)
CLV clear overflow (maak de O-vlag 0)
Afhankelijk of een vlag 0 of 1 is kan er gesprongen worden naar een ander adres. We hebben de volgende mogelijkheden:
BCC spring als C-vlag 0 is BVC spring als V-vlag 0 is
BCS spring als C-vlag 1 is BVS spring als V-vlag 1 is
BNE spring als Z-vlag 0 is BPL spring als N-vlag 0 is
BEQ spring als Z-vlag 1 is BMI spring als N-vlag 1 is

DEEL 3:
Eindelijk is het nu zover: Na de droge theorie uit voorgaande afleveringen, gaan we nu beginnen programma's te schrijven.
Eerst gaan we echter de monitor van R.Heuvel laden en gebruiksklaar maken. Ik ga hierbij uit van archiefbandje 3 of 9. Bandje 9 bevat een bijgewerkte versie. Deze laadt met *LOAD"MRH 7 SRC"8200. Na laden, als bekend, ?18=≠82 END RUN Op de vraag 'schakelaar omzetten' gewoon return toets. Het programma assembleert dan en geeft zijn assemblerlisting zoals hij zichzelf heeft weggeschreven naar het door u gekozen adres. Daarna verschijnt de : u zit dan ín de monitor. Ga er nu uit met 'ESC'. Wie de monitor in de geheugenkaart gezet heeft vanaf bv. ≠7C00-≠7FFF, kan hem nu met Write-Protect op slot doen.
Heeft u geen kaart en (bv.) ook niet 'gestapeld', dan staat de monitor waarschijnlijk op ≠3800-≠3BFF. U kunt deze geassembleerde versie dan het beste direct even op de band zetten, dan bent u volgende keer sneller klaar. Dit gaat (in dit geval) met *SAVE"MRH 7 MC"3800 3C00 3AD5. De geassembleerde versie 7 is maar 1K lang. Als alles zover klaar is, dan is de monitor aan te roepen met LINK≠3AD5 Met de JUMPER van Bouma (nu of straks bv. met MRH.
De MC (Machine Code) versie, die direct naar de geheugenkaart kan, naar ≠7C00, staat reeds op bandje 9. In dit geval LINK≠7ED5.
We verlaten de monitor nu met ESC en gaan aan de slag:
Type in (gewoon in het lage geheugen ≠2900 dus)

```
  10 P=≠2800       (de P verstaat de Atom als beginadres waarop een
                          program geassembleerd moet worden)
  20[LDX@0         (@ is hier 'apestaart'. Hier staat: laadt X register
                         onmiddellijk (immidiate) met een 0)
  30 INX           (Increment, d.i. vermeerder X met 1)
  40 JMP ≠2802     (Jump, d.i. spring naar adres ≠2802)
  50];END           N.B. Alle spaties zijn hier getypt voor de duide-
                         lijkheid. U kunt ze zelf het beste weglaten.
```

We typen nu RUN en krijgen van de ATOM een korte disassemblerlisting.
We roepen nu de Monitor op met LINK enz. en typen achter de : in
:2800 dan Control N dan T, dus :2800 CntrN T (returntoets)
De monitor gaat nu vanaf ≠2800 de Codes en Registers 'Traceren' en doet dit, door de ControlN, in de PAGE MODE, dus steeds een pagina. De volgende pagina krijgt u door indrukken van 'n toets. Dit traceren kunt u stoppen met ESC. Dat geeft de : terug. Met nogmaals ESC gaat u dan de Monitor uit.
Bij het Traceren ziet u de 'waarde' van register X ónder de X staan. U

ziet, dat na elke instructie INX deze waarde 1 hoger wordt, daarna wordt er weer terugge-'jumpt' naar ≠2802.
N.B. Zolang u nu de BREAKtoets niet gebruikt, blijft de ControlN d.i. dus de Pagina Mode erin staan. Hoeft u niet meer te doen. Nogmaals van begin af kijken met de Monitor gaat dus met :2800T
U ziet hierbij, dat in eerste instantie X geladen wordt met 0. Dit komt door de instructie van regel 20. Dit heet 'immidiate' adressering met als kenmerk de Apestaart.
ONTHOUD: LDA@ betekent: Laad X met getal achter @en heet 'immidiate' (immiediejut) adressering.
Bekijken we de Tracering nog eens goed. (In het Engels TRACE (trees).)
Links dus de PC, de Program Counter.
De PC geeft het adres waar de code staat, die in deze regel behandeld wordt. Daarna de Registers A, X en Y
Het A Register heet in het spraakgebruik de 'Accu'. We zeggen altijd dus bv. 'laad Accu' en niet 'laad A-Register'. Wat hetzelfde is.
Welke waarde nu in A en Y staan is in dit program niet interessant.
Geheel rechts zien we de 'Vlaggen', de FLAG's (spreek uit flèks). n.l.
N V B D I Z C, allemaal éénbit-registers, die dus alleen maar + of - kunnen zijn of Hoog/Laag of 1 en 0.
Op het moment dat X geladen wordt met 0 zien we, dat de Z Flag 1 wordt, en alléén dan! De Z Flag heet ZERO-Flag. Zero is het 'nul' in Engels. De Zero-Flag wordt 1 als een Register 0 wordt. Hij wordt weer Zero (0 dus) bij alle andere waardes.
Wanneer u het programma met de Monitor bekijkt, ziet u, dat X steeds één cijfer hoger wordt. Wanneer u het programma lang laat lopen dan verandert er op een gegeven moment weer een Flag. Noteer zelf welke Flag dan 'geset' wordt (1 wordt) en wanneer.
Het grootste getal, dat het A, X en Y Register bevatten kan is ≠FF oftewel 255 (in decimaal). Verder tellen geeft weer 0.

Een Register kan dus geladen worden 'immidiate' met een getal achter @.
We kunnen een register ook laden met een getal, dat 'ergens' op een adres staat, als we bij de instructie dit adres maar opgeven.
B.v. LDX ≠86 Dit heet: ABSOLUTE ADRESSERING.
Neem de proef maar: U vult b.v. ≠86 met ≠3. U kunt dit doen door 'gewoon' te pooken volgens het Manual, dus: ?≠86=≠3 (returntoets).
Met de Monitor gaat het vlugger: U toetst in
:86:3 (ret) N.B. de Monitor verstaat álle getallen als Hexadecimaal.
U gaat nu met ESC de Monitor uit en met L. (=List) in het program, de SOURCE (Bron), dan verandert u regel 20 in 20[LDX ≠86
Na RUN gaat u weer met de Monitor kijken. U ziet dan, dat de uitgangswaarde van X niet meer 0 is maar 3 (≠3 dus in feite, is hier hetzelfde).
Wilt u alleen maar deze eerste (uitgangswaarde) zien, typt u dan
:2800S (ret) Wilt u slechts (b.v.) 4 stappen zien dan
:2800SSSS (ret) Wilt u alleen de assemblerlisting zien, dus zonder de Registers, dan :2800L (ret) Combineren kan ook, bv. :2800SL

Gaan we met ESC terug naar ons oorspronkelijk programma, de Source(Soers)
Op regel 30 staat de instruktie INX (increment X, spr. uit inkriement).
Zo'n instruktie heet een IMPLIED instruktie (implijet). In deze instruktie ligt vast: a) Wat er moet gebeuren en b) impliciet ook waar. Er staat dus: a) Tel 1 op; b) in register X.
Ons programma tot dusverre telt eindeloos éénen op in Register X. We kunnen ook laten optellen tot aan een bep. waarde. Die waarde moeten we opgeven. Het programma vergelijkt dan na élke optelling of die waarde al of niet bereikt is. 'Vergelijken' in Engels is: To COMPARE (kompèr).
Vandaar de instruktie CMP (Compare), en de instruktie CPX (Compare X Re-

gister met Geheugen).

De CPX instruktie vergelijkt de waarde van het X Register met de waarde die achter die instruktie moet worden opgegeven.
Dit kan (alweer) 'immidiate', dus met @ of 'absolute' met ADRES. Dus:
CPX@ #2A   betekent: Verg.Reg.X met getal #2A   en:
CPX #2A   betekent: Verg.Reg.X met getal dat op adres #2A staat.
Het 'Compare' of 'Vergelijken' doet de machine, door de twee te vergelijken waarden van elkaar af te trekken. De waarden vindt hij dan 'gelijk' als het verschil op 0 uitkomt. Niet zo dom geredeneerd. 'Gelijk' in Engels is EQUAL (Iekwel).
Bij EQUAL moet het program iets anders gaan doen dan blijven optellen. Het program moet gaan 'vertakken' ergens anders heen. 'Tak in Engels en Frans is BRANCH.

Vandaar de instruktie BRANCH ON EQUAL afgekort BEQ.
Wat gebeurt er nu allemaal in feite:
-Bij de instruktie CPX wordt de inhoud van het X register vergeleken met de opgegeven waarde, immidiate of absolute.
-De waarden zijn gelijk, als het verschil 0 is. Wanneer deze waarde 0 is dan wordt de Z Flag 'geset', dus 1 gemaakt.
De instruktie BEQ leest eigenlijk alleen de Z Flag, en zogauw deze 1 wordt laat hij het program vertakken.
WAARHEEN branchen (vertakken)?
Moeten we weer opgeven achter de instruktie BEQ. Dit kan a) door achter de instruktie een adres op te geven, b) door achter de instruktie een label op te geven.

Een LABEL in assembler is b.v. LL10 maar ook AA0
Een label wordt in het program geplaatst door :LL10 Zo'n label(leebel) kan vóór aan een programregel worden gezet, maar ook binnen de regel na een ; (punt/komma).

Wanneer we in een programma laten BRANCHEN naar een LABEL, en dat LABEL staat pas verderop in het program, dan is die BEQ instruktie op dat moment niet uitvoerbaar. Het program moet deze LABEL bij een eerdere run éérst hebben zien staan en het adres ervan onder DIM hebben genoteerd. Een dergelijk program 'met een jump vóóruit' heeft dus bij de eerste keer runnen geen effect en loopt goed bij de twééde RUN.

Demonstreren we eerst de functie van een LABEL met:

```
 5DIMLL1          30:LL0INX
10P=#2800         40JMPLL0
20[;LDX@0         50];E.
```

Disassembleren we dit program (na RUN) met de Monitor (:2800L) dan zie je, dat er t.o.v. Voorbeeld 1 niets veranderd is. Het Label wordt gewoon ingelezen als het adres, waar dit label staat.
Typ b.v. PRINT &LL0 en er verschijnt 2802.
Het voordeel van LABEL's zal duidelijk zijn. Wanneer je in een SOURCE het BEGIN ASSEMBLEERADRES P veranderd wil hebben door deze machine code ergens anders te plaatsen in je geheugen, dan hoef je bij gebruik van LABELS niet alle JMP en BEQ instrukties zelf van nieuwe adressen te voorzien. Het program zoekt zelf wel op waar de labels staan.

Weer terug naar de CPX instruktie.
Achter BEQ vullen we dus in LL0 en als de 'COMPARE' is 'ZERO' dan volgt een 'BRANCH' naar het adres van LL0. Komt dit label LL0 pas verderop in het program dan verschijnt in de disassemblerlisting de boodschap 'out of range' wat zoveel betekent als 'ik lig eruit'. Een feitelijke errormelding is dit niét. Het disassembleren gaat gewoon door en aan het eind

zijn de plaatsen van de labels genoteerd. Bij een tweede keer RUNnen gaat het prima. Dit twéé keer runnen kun je het programma zélf laten doen (R20). Bij 'sprongen vóóruit' zullen we dit verder altijd doen.

```
  10DIMLL2
  20GOS.a;GOS.a
  30E.
 100aP=≠2800;[
 110LDX@0
 120:LL0INX
 130CPX@≠10
 140BEQLL1
 150JMPLL0
 160:LL1BRK        (Break is STOP, net als bij boksen)
 170];RETURN
```

Wat gebeurt hier allemaal. We zullen u nog één keer helpen: Met DIMLL2 worden aan het eind van het program 2 geheugenplaatsen ge'dimensioneerd' (gereserveerd) voor 2 labels genaamd LL0 en LL1. Het programma 'runt' 2x i.v.m. de jump-vóóruit, zie 140 en 160. Bij de eerste keer 'RUN' ziet u 'OUT OF RANGE' verschijnen; de tweede keer niet meer.

110 zet een 0 in Register X
120 maakt dit getal 1 hoger
130 kijkt of er al 10 staat (Decimaal 16)
140 zo niet; slaat 140 over en jumpt met 150 terug naar 120
160 zo ja; branch't (jumpt) naar 160 en stopt.

Nu weer de Monitor in en dan disassembleer vanaf 2800 (:2800L). Het program heeft daar 6 regels en gebruikt 2800 t/m 280A = 10 geheugenplaatsen.

De afkortingen LDX, CPX enz. noemen we MNEMONIC's (Maemoniks). Het MNEMONIC LDX is dus de INSTRUKTIE 'laad Reg.X met ..' en de MACHINE CODE voor deze instruktie is A2. Wanneer u dus de instruktie LDX intypt, dan maakt de machine hiervan de Machine Code A2 want de computer begrijpt alleen getallen. Dus: alle getallen(waarden), alle adressen en alle instrukties zijn in machine code g e t a l l e n die op een adres (getal) staan. OMGEKEERD (en daar gaat het om) kan élk getal in de machine zijn: een waarde(getal), een adres of een instruktie.

Wat met een getal bedoeld wordt begrijpt de machine alleen uit de v o l g o r d e van de getallen. Komt de machine dus het getal A2 tegen, dan zal hij dit begrijpen als LDX@ (load X immidiate). Staat daarachter wéér A2 dan vat hij dit als de 'waarde' op waarmee X moet worden geladen.

We kunnen dit laten zien: Type in (in de Monitor) :2800:A2 A2 (ret) (Let op de spatie). Daarna 2800L (ret) U ziet dan LDX@≠A2
Probeer daarna :280A:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (ret)
:2800L (ret)
Het is dus duidelijk; de Code voor BRK is 0.
Wanneer u deze codes een beetje kent kunt u met de Monitor d i r e c t in machinecode veranderingen aanbrengen in een program. Probeer maar eens (als oefening) :2808:0C (ret) :2801:9 (ret)
:280C:4C 02 28 (ret) :2800L (ret)
U ziet dan de PC (Program Counter) braaf van 2807 naar 280C springen en direct weer terug naar 2802.
Probeer een paar CODES te onthouden, zoals hierboven 4C (jump naar ..).

We gaan weer terug naar ons programma met ESC en LIST.
Dit programma is dus een gemengd BASIC/ASSEMBLER programma. Wanneer we RUN intypen assembleert het zichzelf naar ≠2800. RUN is een BASIC STATEMENT. Wanneer je het geassembleerde program d i r e c t wil 'INITIEREN' (op gang brengen) dan doe je dit met LINK≠2800 Eerst moet dan de instruktie BRK in regel 160 veranderd worden in RTS (Return from Subroutine). U doet dat natuurlijk met de Monitor; de BRK staat op adres 280A en de CODE voor RTS is 60. Dus simpel :280A:60 (ret) ESC LINK≠2800
Je ziet dan nix maar in het X Reg. staat dan echt 10. Ga maar kijken.

Wanneer u een paar CODES kent, dan is het vaak simpeler om in een HEX DUMP te kijken en/of te veranderen. Type maar eens (in de monitor) :2800 (ret) geeft :2800: A2 (weer ret) geeft 00 E8 E0 10 F0 03 4C (ret) geeft 2802: 0C 28 60 60 4C 02 28 00 (en elke ret geeft volgende 8 codes).
U bent nu deskundig en ziet direct staan: 4C 02 28 (jump naar ≠2800)
N.B. ADRESSEN zijn doorgaans 2 bytes lang. De notering gaat in de machine altijd met éérst het LOW BYTE en daarna het HIGH BYTE.

Terug naar onze cursus: Alle instrukties, die tot dusverre genoemd zijn, gelden netzogoed voor het Y Reg. Dus LDY@0, INY en CMY
-Probeer nu eens zelf als oefening, een soort wachtlus te bedenken waarbij ≠FF x ≠FF maal gewacht moet worden. Gebruik het X én het Y Reg.

Nóg een instruktie zal ik hierbij uitleggen. Dat is de instruktie om het programma beschaafd te laten stoppen. Dat is die RTS instruktie (Return from Subroutine). Hiermede springt het programma terug naar BASIC. Je moet je het in het begin ongeveer zó voorstellen: Als je zegt LINK≠2800 dan roep je eigenlijk een subroutine aan. Nu weten we uit BASIC al, dat je een subroutine af moet sluiten met een RETURN. En zo doe je dat ook in machinetaal. 'LINKen' van een program, dat eindigt met BRK geeft een ERROR. Dit komt omdat de BASIC-INTERPRETOR (spreek uit intùrpr'tr) zelf een BRK schrijft, wanneer hij een fout ontdekt en springt daarmee naar de ERROR-HANDLER (het fouten-afwikkelsysteem). Als je nu zélf met LINK naar een subroutine springt, waar al een BRK instaat, dan schrijft hij er nog een BRK achteraan en springt naar de Error Handler. Die maakt er een ERROR van en -omdat er eigenlijk geen fout is- verzint die er maar een nummer bij, b.v. 12.

SAMENVATTING deel 3:

IMMIDIATE ADRESSERING	LDA@≠2A , CMP@≠7C
ABSOLUTE ADRESSERING	LDA≠B002 , CPY≠8000
IMPLIED ADRESSERING	INX , DEX , INY , DEY (deekriement Y)
RELATIEVE ADRESSERING	BEQLL0 , BNELL4 , BCCMM3

(BNE = BRANCH IF <u>not</u> EQUAL   BCC = BRANCH als <u>C</u>arryFLAG = 0)
Verder:-Bij sprongen vóóruit moeten we 2x assembleren -Een programma sluiten we af met RTS -De Monitor is een schitterend hulpmiddel.

De WACHTLUS ≠FF x ≠FF maal zou er als volgt uit kunnen zien: (Probeer zelf alle overbodige dingen eruit te halen. Bedenk daarbij dan ≠FF+1=0. Herinner ook,dat als X of Y 0 wordt, de Z Flag geset wordt, dus 1 wordt).
N.B. Een Flag 0 maken heet 'resetten' of 'CLEAR' b.v. CarryFlag.

```
 10DIMLL2                  Reserveer plaats voor 2 labels
 20GOS.a;GOS.a             Doorloop program 2x
 30E.
100aP=≠2800                Beginadres ass. code
110[LDX@≠00                Load Reg.X met getal 0
120:LL0LDY ≠00             Load Reg.Y met getal 0
130:LL1INY                 Y=Y+1
140CPY@≠FF                 Is Y gelijk aan ≠FF ?
150BNELL1                  Zo niet, spring naar 130
160INX                     X=X+1
170CPX@≠FF                 Is X gelijk aan ≠FF ?
180BNELL0                  Zo niet, spring naar 110
190RTS                     Keer terug vanwaar je kwam
200];RETURN
```

S U C C E S !!   Leendert Bijnagte

PEEKO

INLEIDING:
De bedoeling van dit simulatie programma is de 'beginner' inzicht te geven in hoe de microprocessor werkt. Dit programma laat vooral zien dat een bep. 'BYTE' in het geheugen van de processor 3 functies kan hebben:
a) Een gecodeerde instructie, b) Een gedeelte van het adres van een bep. geheugenplaats, c) Gegevens die de processor verwerkt of modificeert tijdens de uitvoering van een programma.
BYTE: Een groep van 8 opeenvolgende bits die als één eenheid wordt verwerkt en één geheugenplaats inneemt.
De gesimuleerde processor heeft 50 geheugenplaatsen. Elk van deze geheugenplaatsen kan een enkel decimaal getal bevatten. De inhoud van het geheugen en van de accumulator is continu zichtbaar, evenzo de toestand van de 'carry' en van de 'input en output' poorten.
CARRY: Toestandbit in het toestandsregister van de centrale verwerkings eenheid (CPU) dat aangeeft of het resultaat van een rekenkundige bewerking in de ALU (Schakeling in de processor waarin de rekenkundige en logische beweringen worden uitgevoerd.) groter is dan het grootste getal dat kan worden weergegeven.
De instructieset van de PEEKO is beperkt tot 10 instructies, die veelal overeenkomen met de instructies van de 6502. Maar deze 10 instructies kunnen, indien gewenst, veranderd worden.
Het opsporen en verwijderen van fouten (debuggen) wordt aanzienlijk vergemakkelijkt door de mogelijkheid het programma stap voor stap uit te voeren. Hierbij geeft de cursor (program counter) de volgende instructie aan die moet worden uitgevoerd.
PROGRAM COUNTER: Een register, dat het geheugenadres van de volgende instructie die in het computerprogramma uitgevoerd moet worden, bevat.
Als u eenmaal de PEEKO-computer en al zijn mogelijkheden onder de knie hebt is het eenvoudig om echte machinetaal programmas te schrijven voor de 6502.

WAT DOEN MICROPROCESSORS:
1) Een microprocessor gebruikt een aantal geheugenplaatsen. Iedere geheugenplaats bevat één byte. Voor de PEEKO-computer is iedere byte één enkel decimaal getal.
2) Elke geheugenplaats heeft een eigen adres. Bij de PEEKO-computer wordt het adres aangegeven met twee decimale getallen. De computer leest deze uit van links naar rechts en rij voor rij. Deze adressen zijn bv.: 00,01, 02,.....,48,49. De inhoud van zo'n adres kan bv. voorstellen: a) Een instructie. De PEEKO kent 10 verschillende instructies 0 t/m 9. b) Een gedeelte van een ander adres. Er zijn 2 bytes nodig om een adres aan te geven. c) Gegevens die de processor kan verwerken of modificeren tijdens de uitvoering van een programma.
3) De processor zelf heeft registers die gebruikt worden tijdens de uitvoering van een programma. De PEEKO heeft in tegenstelling tot de 6502 maar één register en wel de accumulator (accu).
REGISTER: Een hulpgeheugen of deel van een geheugen dat meestal de capaciteit heeft van een computerwoord en dat een speciale functie heeft in de computer.
4) Enkele speciale adressen kunnen gebruikt worden om informatie uit te wisselen met de 'buitenwereld', de zgn. input- en outputpoorten. In de PEEKO heeft de inputpoort het adres 98 en de outputpoort het adres 99.

PEEKO MONITOR:
Als u het PEEKO programma geladen en 'gerund' heeft, dan hebt u toegang

tot de PEEKO-monitor.
MONITOR: Programmatuur of apparatuur die de juiste uitvoering van een programma bewaakt en controleert, gewoonlijk d.m.v. stelselmatige checks met een zgn. diagnostic routine die antwoord geeft op vragen omtrent de programmatuur. Of eenvoudiger: Programma dat de basissysteemcommando's interpreteert en uitvoert. U kunt de monitorcommando's invoeren middels de hierna volgende (code)-toetsen.

0 - 9 Elke cijfertoets verandert de inhoud van de geheugenplaats die door de cursor wordt aangegeven. De cursor gaat dan naar de volgende geheugenplaats. Bovendien hoort u een piepje dat aangeeft dat de ingevoerde gegevens geaccepteerd zijn. Van elke ingevoerde instructie wordt ook de mnemonic getoond (in de linkerbovenhoek van het scherm). Dit eventueel met de daarvoor benodigde argumenten.

G (GO) Laat de PEEKO het programma dat in het geheugen staat uitvoeren, beginnend op de geheugenplaats 0. Dit gebeurt totdat er in het programma een BRK-instructie voorkomt of totdat u de E-toets indrukt.

E (ESCAPE) Als de PEEKO in de 'RUN'-status is stopt het E-commando de uitvoering van het programma en de PEEKO wordt teruggebracht in de READY-status.

SPATIE Door de spatie-toets te gebruiken wordt het programma stap voor stap uitgevoerd en elke uit te voeren instructie wordt in de linkerbovenhoek van het scherm getoond.

F (FAST) Hetzelfde als G, alleen wordt het programma sneller uitgevoerd.

↔ ↕ Controle toetsen om de cursor op iedere willekeurige positie op het scherm te plaatsen.

O (ORIGIN) Plaatst de cursor op geheugenplaats 0.

S (SAVE) Door het S-commando kunt u een PEEKO-programma in het 'graphics' geheugen van de ATOM laden. Dit commando moet gevolgd worden door een decimaal getal van 0 t/m 9 om het programma te identificeren. Als u over niet meer dan 3K 'graphics'-geheugen beschikt kunt u alleen de getallen 0 t/m 3 gebruiken.

L (LOAD) D.m.v. het L-commando gevolgd door een decimaal getal kunt u een programma uit het geheugen van de ATOM laden in de PEEKO-computer. (Zo kunt u bv. ook de demonstratie programma's: ADD,COPY,FACTOR eerst laden in het 'hoge' geheugen van de ATOM en vervolgens middels dit commando laden in de PEEKO-computer.)

I Biedt de mogelijkheid de instructie set van de PEEKO te veranderen. Naast de standaard set van 10 instructies beschikt de PEEKO over een extra set van 10 instructies. Elke standaard instr. kan vervangen worden door een instr. uit de extra set. Als de PEEKO in deze mode (I) is kunt u door het 'S' commando teruggaan naar de standaard instr. set.

P (PRINT) Door dit commando kunt u het programma dat zich in het geheugen van de PEEKO bevindt via de printer laten afdrukken. Als u gebruik maakt van een Acorn GP-80 printer gebeurdt dit met extra grote letters. Voor andere printers is misschien een verandering van de programma regels 9400 tot 9410 noodzakelijk.

N (NEXT) Met dit commando kunt u een programma regel voor regel 'listen', waarbij iedere mnemonic op de regel onder 'RUN' op het scherm getoond wordt.

> Hiermee kunt u een programma en de toegepaste instr. set op cassetteband zetten. Dit commando vraagt eerst om een door u in te voeren filenaam.

< Hiermee kunt u een programma van cassetteband laden. Ook dit commando vraagt eerst om een door u in te voeren filenaam.

INSTRUCTIES:
Elke instr. heeft een 'naam' van 3 of 4 letters, deze 'naam' wordt mne-

monic genoemd. De mnemonic is doorgaans een afkorting van de beschrijving van wat de instr. doet. De verschillende instr. hebben specifieke gegevens nodig die achter de instr. moeten worden opgegeven. Sommige instr. hebben 2 bytes nodig. Om het adres aan te geven van de geheugenplaats waar de instr. betrekking op heeft. De instr. 'STA' (bewaar de inhoud van de accu) moet bv. gevolgd worden door het adres van de geheugenplaats waarin de informatie moet worden opgeborgen.

INSTRUCTIE STA ← 2 0 4 → ADRES 40

BELANGRIJK: Bij de PEEKO-computer moet de laagste byte van een adres eerst opgegeven worden. De instr. voorafgaand aan de adrescode interpreteert dit echter als de normale schrijfwijze. VOORBEELD: Het adres '40' moet aan de PEEKO worden opgegeven als '04'.
Sommige instr. worden 'onmiddellijk' gevolgd door een enkele byte, die dan de data voorstelt die door de instr. moet worden gebruikt. VOORBEELD: De 'laad de accu direct met' (LDA#) instr. moet worden gevolgd door het getal dat moet worden geladen.
Sommige instr. hebben helemaal geen data nodig. VOORBEELD: De instr. 'clear carry' zet de waarde 0 in de 'carry flag'.(De carry flag is een register dat ongeveer het zelfde doet als bv.: 'het 1 onthouden bij optel sommetjes')

INSTRUCTIE-SET:
De standaard instructie-set van de PEEKO is als volgt:

0	BRK	Stop het programma en ga door naar de monitor.
1	LDA xx	Zet de inhoud van adres xx in de accu.
2	STA xx	Zet de inhoud van de accu in de geheugenplaats met adres xx.
3	CLC	Geef de carry flag de waarde 0.
4	ADC xx	Tel de inhoud van geheugenplaats xx, met inachtneming van de carry, op bij de inhoud van de accu.
5	DEC xx	Verlaag de inhoud van adres xx (met 1).
6	INC xx	Verhoog de inhoud van adres xx (met 1).
7	LDA #x	Laad de accu 'onmiddellijk' met het getal (de waarde) x.
8	JNE xx	Spring naar het adres xx als het aan de instr. voorafgaande resultaat (van een bewerking) niet gelijk is aan nul.
9	JMP xx	Spring naar adres xx.

PROGRAMMA'S:
Een programma bestaat uit een opeenvolging van instr. die door de processor een voor een uitgevoerd worden en wel beginnend bij het laagste adres. Dit gebeurt totdat de processor een BRK instr. tegenkomt. De 'spring' instr. is ontworpen om deze volgorde te kunnen veranderen en dus de processor zijn volgende instr. te laten halen van een achter de 'spring' instr. nader gespecifixeerd adres.

HET PROGRAMMEREN VAN DE PEEKO:
Het hierna volgende laat zien hoe u de instr. van de PEEKO kunt gebruiken. D.m.v. een demonstratie programma zal het gebruik van de meeste instr. aan u worden duidelijk gemaakt. Elk programma dat behandeld wordt zal worden opgeschreven als een serie mnemonics, gevolgd door de bijbehorende machinecode en in de meeste gevallen door een tekening van het beeldscherm.
VOORBEELD 1: Het optellen van twee getallen.

plaats	mnemonic	code
00	LDA 40	1 0 4
03	CLC	3
04	ADC 41	4 1 4
07	STA 42	2 2 4
10	BRK	0

Begin op plaats 0,0 en typ de code in. Let erop dat iedere instr. op het scherm verschijnt onder het woordje 'RUN'. Typ vervolgens de twee getallen in die moeten worden opgeteld op de plaatsen 40 en 41. Gebruik om op die plaatsen te komen de cur-

sor-controle toetsen. Als u het programma 'stap voor stap' wilt laten 'draaien' gebruik dan de spatie toets. Wanneer u nu de spatie toets opnieuw indrukt wordt de eerste instr. (LDA 40) uitgevoerd. Deze instr. zet de inhoud van geheugenplaats 40 in de accu. De volgende instr. (CLC) maakt de inhoud van de carry 0. De ADC instr. telt de inhoud van adres 41 op bij de inhoud van de accu (met 0 carry) en verandert de inhoud van de accu in het resultaat van de optelling. De STA instr. zet vervolgens dit resultaat (dus de huidige inhoud van de accu) in geheugenplaats 42. Tenslotte stopt de BRK instr. het programma. Als u voor dit optel programma getallen kiest die groter zijn dan 10 dan zal adres 42 alleen het laatste digit (cijfer) bevatten en wordt de carry op 1 gezet. Als u voorbeeld 1 'gerund' hebt, dan ziet u het hierna volgende schermbeeld:

```
 READY   INPUT PORT    OUTPUT PORT
 PROG      98 0           99 0
 RUN
 LDA 40       CARRY 0 ZERO 0 ACC 7
 0  1  0  4  3  4  1  4  2  2  4
 1  0  0  0  0  0  0  0  0  0  0
 2  0  0  0  0  0  0  0  0  0  0
 3  0  0  0  0  0  0  0  0  0  0
 4  3  4  7  0  0  0  0  0  0  0
    └ + =┘
    0  1  2  3  4  5  6  7  8  9
```

VOORBEELD 2: Het optellen van 2 uit 2 digits bestaande getallen.
Dit programma telt de getallen op die op de geheugenplaatsen 40,41 en 42, 43 en zet het resultaat in de geheugenplaatsen 44,45,46.

```
plaats mnemonic code
00     CLC      3
01     LDA 41   1 1 4
04     ADC 43   4 3 4
07     STA 46   2 6 4
10     LDA 40   1 0 4
13     ADC 42   4 2 4
16     STA 45   2 5 4
19     LDA@0    7 0
21     ADC 20   4 0 2
24     STA 44   2 4 4
27     BRK      0
```

Het scherm voor voorbeeld 2

```
READY   INPUT PORT    OUTPUT PORT
PROG      98 0           99 0
RUN
CLC          CARRY 0 ZERO 0 ACC 1
0  3  1  1  4  4  3  4  2  6  4
1  1  0  4  4  2  4  2  5  4  7
2  0  4  0  2  2  4  4  0  0  0
3  0  0  0  0  0  0  0  0  0  0
4  7  6  6  1  1  3  7  0  0  0
   └  +     =     ┘
   0  1  2  3  4  5  6  7  8  9
```

In dit programma wordt een nieuwe instr. gebruikt. Dit is de LDA@ instr. Deze instr. 'laadt de accu onmiddellijk' met de byte die de instr. volgt. In dit geval 0.

PROBLEEM 1:
Schrijf een programma met gebruik van de instr. LDA,STA,ADC en CLC dat de getallen optelt die staan in de geheugenplaatsen 40, 41 en 42 en het resultaat, dat bestaat uit 2 digits, in de geheugenplaatsen 43 en 44 zet.

VOORBEELD 3: Aftel programma.
Het volgende programma trekt van de inhoud van geheugenplaats 40 telkens 1 af. Dit gebeurt in een onafgebroken lus. LET OP!!!! 0-1=9

```
plaats mnemonic code
00     DEC 40   5 0 4
03     JMP 00   8 0 0
```

In dit programma komen 2 nieuwe instr. voor nl. DEC en JMP. DEC trekt één af van de inhoud van een gespecificeerde geheugenplaats, in dit geval 40. JMP laat het programma verder gaan op een geheugenplaats die achter de instr. gespecificeerd wordt, in dit geval 00. Deze

instr. komt overeen met de GOTO instr. in BASIC. Omdat dit programma een continu lus is, moet E worden ingetypt om uit de 'RUN-modus' te ontsnappen. Door 'F' in te typen inplaats van de spatie toets zal de PEEKO het programma uitvoeren zonder te stoppen na elke instr.

VOORBEELD 4: Tel terug vanaf 99.

Dit programma telt terug naar 0 vanaf een getal dat in de geheugenplaatsen 40 en 41 staat. Bij 0 stopt het programma.

plaats	mnemonic	code
00	DEC 41	5 1 4
03	JNE 00	9 0 0
06	DEC 40	5 0 4
09	BRK	0
40		9
41		9

Tekening van het scherm voor dit programma.

```
READY   INPUT PORT    OUTPUT PORT
PROG      98 0           99 0
RUN
DEC 41         CARRY 0 ZERO 0 ACC 1
0  5  1  4  9  0  0  5  0  4  9
1  0  0  0  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0  0
4  9  9  0  0  0  0  0  0  0  0
  counter
   0  1  2  3  4  5  6  7  8  9
```

Als u dit programma uitvoert let er dan wel op dat als de inhoud van geheugenplaats 41 nul wordt, de inhoud van de 'zero-flag' 1 wordt. De inhoud van de zero-flag wordt 1 als de uitkomst van een wiskundige bewerking nul is. In alle andere gevallen is de inhoud van de zero-flag 0.

ZERO-FLAG: Een flip-flop, die de logische waarde 1 krijgt indien het resultaat van een instr. de waarde nul heeft.

De JNE-instr. doet het zelfde als de JMP-instr. behalve dan dat de 'sprong' veroorzaakt door de JNE-instr. pas plaats vindt als het resultaat van de voorafgaande bewerking 0 is. In dit programma wordt er t.g.v. deze instr. dan ook tot nul geteld.

De INC-instr. is ongeveer gelijk aan de DEC-instr., alleen telt deze instr. 1 op bij de inhoud van het adres dat achter de instr. gespecificeerd is. Ook deze instr. zet de inhoud van de zero-flag op 1 als het resultaat nul is. Door de instr. DEC te vervangen door de instr. INC en de inhoud van de adressen 40 en 41 op nul te zetten, kan het vooraf gaande programma van nul tot 100 tellen.

PROBLEEM 2:

Schrijf een programma dat telt van 99 naar nul en van nul naar 100, dit in een continu lus.

NIEUWE INSTRUCTIES:

De instr. die tot nu toe besproken werden behoorden tot de standaard instr.-set. Maar er zijn 10 nieuwe instr. mogelijk die de oude kunnen vervangen. Het I-commando van de PEEKO-monitor vraagt naar het code-getal van de instr. die vervangen moet worden en naar het code-getal van de nieuwe instr. O.a. door deze mogelijkheid kunt u de PEEKO voor allerlei taken programmeren, dit ondanks de geringe programma omvang. De werking van de meeste van de nieuwe instr. is eenvoudig te verklaren omdat deze instr. vrijwel gelijk zijn aan de instr. van de standaard set.

SEC	Zet 1 in de carry
SBC xx	Trek met inachtneming van de carry de inhoud van adres xx af van de inhoud van de accu.
JCC xx	Spring naar adres xx als de carry nul is.
JCS xx	Spring naar adres xx als de carry 1 is.
JEQ xx	Spring naar adres xx als de zero-flag 1 is.
DECA	Trek van de inhoud van de accu 1 af.
INCA	Tel bij de inhoud van de accu 1 op.

CMP x   Vergelijk de inhoud van de accu direct met x. Als beide gelijk zijn, maak dan de zero-flag 1, zo niet, maak dan de zero-flag 0.

LDA (xx) Laad de accu met de inhoud van het adres dat wordt aangegeven door xx en xx+1.

STA (xx) Zet de inhoud van de accu in de geheugenplaatsen met het adres dat wordt aangegeven door xx en xx+1.

De laatste 2 instr. zijn indirecte 'load- en store' instr. Omdat deze instr. wat ingewikkelder zijn dan de andere instr. zal hierna in voorbeeld 6 uitgebreid aandacht besteed worden. De volgende 5 programma's gebruiken sommige van genoemde mogelijkheden. Als u deze instr. gebruikt moet eerst de instr.-set veranderd worden. Dit wordt voor ieder voorbeeld apart aangegeven.

VOORBEELD 5: Het aftrekken van twee, twee-cijferige getallen.

De instr.-set verandert als volgt: 3 SEC   4 SBC

plaats	mnemonic	code	plaats	mnemonic	code	plaats	mnemonic	code
00	LDA 41	1 1 4	07	STA 45	2 5 4	16	STA 44	2 4 4
03	SEC	3	10	LDA 40	1 0 4	19	BRK	0
04	SBC 43	4 3 4	13	SBC 42	4 2 4			

Dit programma berekent het resultaat van de opgave: Trek de inhoud van geheugenplaats 42,43 af van de inhoud van de geheugenplaatsen 40,41 en zet het resultaat in de geheugenplaatsen 44,45. Als de carry 1 is betekent dat dat het resultaat niet negatief is. Is de carry nul dan betekent dat,dat de inhoud van de geheugenplaatsen 42,43 groter was dan de inhoud van de geheugenplaatsen 40,41 en dus dat de uitkomst negatief is.

READY   INPUT PORT   OUTPUT PORT
PROG   98 0   99 0
RUN
LDA 41   CARRY 1 ZERO 0 ACC 6

	0	1	2	3	4	5	6	7	8	9
0	1	1	4	3	4	3	4	2	5	4
1	1	0	4	4	2	4	2	4	4	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	6	7	1	1	5	6	0	0	0	0

VOORBEELD 6: Het indirect adresseren.

Verander hiervoor de instr.-set als volgt: 1 LDA (2 STA (

Dit programma maakt gebruik van de zgn. 'indirect adresseren'. Als de inhoud van de geheugenplaatsen 40 en 41 is: 0 2   en de inhoud van de geheugenplaats 20 is: 8   dan zal t.g.v. de instr. LDA (40) de accu worden geladen met de inhoud van geheugenplaats 20. De inhoud van de accu wordt nu dus 8. LDA (40) betekent dus: Laad de accu met de inhoud van de geheugenplaats waarvan het adres staat op de geheugenplaatsen (in dit geval): 40 en 40+1.

plaats	mnemonic	code
00	LDA (40)	1 0 4
03	STA (42)	2 2 4
06	BRK	0
40		0
41		3
42		0
43		2

Als u dit korte programma draait zult u zien hoe indirect adresseren gebruikt kan worden om de inhoud van een bepaald adres naar een ander adres te brengen. In dit geval van adres 30 naar adres 20.

VOORBEELD 7: Het kopieren van een tabel bestaande uit 10 cijfers.

Het programma dat gebruikt wordt voor dit voorbeeld staat ook op cassette en wel onder de file-naam 'COPY'. Het kan in de PEEKO geladen worden door het commando ' ' te gebruiken. Verander de instr.-set als volgt:

1 LDA (
2 STA (

plaats	mnemonic	code
00	LDA (40)	1 0 4

READY   INPUT PORT   OUTPUT PORT
PROG   98 0   99 0
RUN
LDA (40) CARRY 0 ZERO 0 ACC 8

```
03      STA (42) 2 2 4
06      INC 40   6 0 4
09      INC 42   6 2 4
12      JNE 00   9 0 0
15      BRK      0
40               0
41               2
42               0
43               3
```

0	1	0	4	2	2	4	6	0	4	6
1	2	4	9	0	0	0	0	0	0	0
2	1	2	3	4	5	6	7	8	9	0
3	0	0	0	0	0	0	0	0	0	0
4	0	2	0	3	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

(Copy from here ↑ (row 2) — ↑ to here (row 3))

Dit programma laat zien hoe u op eenvoudige wijze, door gebruik te maken van het indirect adresseren, toegang kunt krijgen tot een data tabel. Het programma kopieert een aantal data. Het begin van deze data rij wordt aangegeven op de adressen 40 en 41. De plaats waarnaar de data gekopieerd moeten worden wordt aangegeven op de adressen 42 en 43. De data staat op de adressen 20 t/m 29 en moeten gekopieerd worden naar de adressen 30 t/m 39.

VOORBEELD 8: Het optellen van 2 rijen data.

Ook dit programma staat op cassette onder de file-naam 'ADD'. Verander de instr.-set als volgt: 1 LDA (2 STA (7 STA Het programma telt 2 rijen data, die al in het geheugen zitten, op en zet de uitkomst in het geheugen. Omdat de geheugenruimte van de PEEKO beperkt is worden 2 rijen data uit het programma zelf opgeteld.

```
plaats mnemonic code
00      CLC       3
01      LDA (48) 1 8 4
04      STA 43   7 3 4
07      LDA (46) 1 6 4
10      ADC 43   4 3 4
13      STA (44) 2 4 4
16      DEC 48   5 8 4
19      DEC 46   5 6 4
22      DEC 44   5 4 4
25      JNE 01   9 1 0
28      BRK

44               9
45               3
46               9
47               1
48               9
49               0
```

```
READY  INPUT PORT   OUTPUT PORT
PROG     98 0          99 0
RUN
CLC         CARRY 0 ZERO 0 ACC 5
```

0	3	1	8	4	7	3	4	1	6	4
		+	+	+	+	+	+	+	+	+
1	4	3	4	2	4	4	5	8	4	5
2	6	4	5	4	4	9	1	0	0	0
		=	=	=	=	=	=	=	=	=
3	0	5	2	7	1	8	0	0	0	9
4	0	0	0	1	0	3	0	1	0	0
	0	1	2	3	4	5	6	7	8	9

Ook dit programma maakt uitgebreid gebruik van indirecte adressering.

VOORBEELD 9: Zoek de factor van een getal.

Het programma staat ook op cassette onder de file-naam 'factor'. (In dit geval betekent 'factor' het grootste gehele deeltal (kleiner dan 10).) Voor dit programma zijn de volgende veranderingen in de instr.set nodig: 3 SEC 4 SBC 8 JCC Het programma berekent de hoogste factor kleiner dan 10, van 2 cijferig getal. Dit getal staat op de geheugenplaatsen 48 en 49

```
plaats mnemonic code   plaats mnemonic code   plaats mnemonic code
00      LDA 48  1 8 4  16      JCC 00  8 0 0  32      STA 28  2 8 2
03      STA 28  2 8 2  19      LDA#0   7 0    35      JNE 16  9 6 1
06      LDA 49  1 9 4  21      SBC 47  4 7 4  38      LDA 20  1 0 2
09      STA 20  2 0 2  24      STA 20  2 0 2  41      JNE 16  9 6 1
12      DEC 47  5 7 4  27      LDA#0   7 0    44      BRK     0
15      SEC     3      29      SBC 46  4 6 4  46              0
                                              47              0
```

Zet het getal waarvan u de factor wil berekenen in de geheugenplaatsen 48 en 49.

```
READY  INPUT PORT    OUTPUT PORT
PROG      98 0           99 0
RUN
LDA 48       CARRY 0 ZERO 0 ACC 5
```

0	1	8	4	2	8	2	1	9	4	2
1	0	2	5	7	4	3	8	0	0	7
2	0	4	7	4	2	0	2	7	0	4
3	6	4	2	8	2	9	6	1	1	0
4	2	9	6	1	0	0	0	8	9	6
					factor ↗			number		
	0	1	2	3	4	5	6	7	8	9

PROBLEEM 3:
Schrijf een programma dat 2 getallen met elkaar vermenigvuldigt m.b.v. de optel-functie.

PROBLEEM 4:
Schrijf een programma voor de PEEKO dat een twee cijferig getal deelt door een getal dat uit één cijfer bestaat. U hebt daarvoor in ieder geval de instr. SBC en SEC nodig en waarschijnlijk ook nog enkele andere. Gebruik hierbij het FACTOR-programma als voorbeeld.

INVOER EN UITVOER POORTEN:
INPUT: Het 'voer' van een centrale verwerkings eenheid dat via een toetsenbord of een extern geheugen wordt overgebracht naar het interne geheugen.
OUTPUT: De uitvoer van data naar randapparatuur of extern geheugen.
Deze 2 poorten, die gebruikt kunnen worden via de geheugenplaatsen 98 en 99 van de PEEKO-computer, komen overeen met de A- en B-poort van het 6522 VIA IC. Dit IC kan in de ATOM geplaatst worden. De ingangspoort 'leest' in principe 4 bits van poort A in en zet deze op geheugenplaats 98 van de PEEKO. Als de invoer groter is dan 10 dan wordt deze teruggebracht tot één cijfer en wordt de carry-flag op één gezet. De uitgaande poort '99', gebruikt de VIA-poort B. Omdat een eventueel aangesloten printer ook het 6522 IC gebruikt, moet u alle andere aan dit IC aangesloten apparatuur afkoppelen en de printer aansluiten als u een afdruk van het programma op papier wil hebben.

ANTWOORDEN OP PROBLEMEN:

PROBLEEM 1

plaats	mnemonic	code
00	CLC	3
01	LDA 44	1 0 4
04	ADC 41	4 1 4
07	STA 44	2 4 4
10	LDA #0	7 0
12	ADC 11	4 1 1
15	STA 43	2 3 4
18	LDA 44	1 4 4
21	ADC 42	4 2 4
24	STA 44	2 4 4
27	LDA 43	1 3 4
30	ADC 11	4 1 1
33	STA 43	2 3 4
36	BRK	0

PROBLEEM 2

plaats	mnemonic	code
00	DEC 41	5 1 4
03	JNE 00	9 0 0
06	DEC 40	5 0 4
09	JNE 00	9 0 0
12	INC 41	6 1 4
15	JNE 12	9 2 1
18	INC 40	6 0 4
21	JNE 12	9 2 1
24	JMP 00	8 0 0

FORTH

WAT IS FORTH?
FORTH is een programmeertaal, die ongeveer 13 jaar geleden door C.H.Moore is uitgevonden. De taal is zeer geschikt voor het besturen van apparatuur m.b.v. een computer. De meeste hogere programmeertalen zijn daarvoor te traag en het schrijven van machinetaal routines is lastig.
Enkele eigenschappen van FORTH (op al deze eigenschappen komen we terug) zijn: - FORTH programma's hebben een hoge executie-snelheid
- FORTH heeft weinig geheugenruimte nodig
- FORTH is uitbreidbaar
- FORTH is interaktief
- FORTH is een gestructureerde taal

DE STACK
Voor de berekeningen gebruikt FORTH een STACK (de parameter-stack). Dit betekent: - alle getallen die worden ingetikt, komen boven op de stack.
- alle operaties halen getallen van de stack en zetten het resultaat van de bewerkingen weer op de stack.
We zullen een voorbeeld bekijken.
We tikken in: 2 4 6 3
De getallen 2,4,6,3 staan nu op de stack, de 3 bovenop en de 2 onderop.
Vervolgens tikken we:+
Nu worden de 6 en de 3 van de stack gehaald en de som (6+3=) 9 weer op de stack gezet. Deze bevat nu dus de getallen 2,4,9.
Typen we nu MAX
Dan worden 4 en 9 van de stack gehaald en hun maximum (=9) wordt teruggezet. De stack bevat nu de getallen 2,9.
We typen nu: -
De getallen 2 en 9 worden van de stackgehaald en hun verschil (2-9=) -7 wordt weer teruggezet.
Als we nu intypen: .
Dan wordt het bovenste element (-7) van de stack gehaald en afgedrukt.
De stack is nu leeg. Kortom, het uitrekenen en afdrukken van 2-MAX(4, 6+3) gaat in FORTH met: 2 4 6 3 + MAX - . (ret) Op het beeldscherm zal dan te zien zijn: 2 4 6 3 + MAX - . -7 OK
Als de opdracht op de juiste manier is uitgevoerd reageert FORTH altijd met "OK".
Van dit voorbeeld kunnen we verschillende dingen leren:
- De stack werkt volgens het "LAST- IN FIRST-OUT" principe. Dit betekent dat het getal, dat het laatst op de stack is gezet, er als eerste weer wordt afgehaald. We kunnen dit vergelijken met een stapel borden!!
- De spatie is in FORTH een scheidingsteken. De verschillende getallen worden alleen door een spatie gescheiden.
- ACORN-FORTH werkt alleen met integer getallen. Er bestaan FORTH-versies die ook met 'FLOATING-POINT' getallen kunnen werken. ACORN-FORTH kent zowel enkele- als dubbele-precissie integers. Op het verschil komen we nog terug; voorlopig beperken wij ons tot de enkele-precisie getallen. Dit zijn 16 bits getallen, zodat de waarde die ze kunnen aannemen ligt tussen -32768 en 32767 (ofwel tussen 0 en 65535, indien we ze beschouwen als getallen zonder teken).
- FORTH is op de manier zoals we hebben voorgedaan te gebruiken alsof het een rekenmachine is. FORTH gebruikt een POST-FIX notatie, die de gebruikers van H.P. rekenmachines bekend zal voorkomen.
We zullen een aantal STACK-OPERATIES op een rijtje zetten:

+	berekent de som van de bovenste 2 elementen.
-	berekent het verschil van de bovenste 2 elementen.
*	berekent het product van de bovenste 2 elementen.

/ berekent het quotient van de bovenste 2 elementen.
MOD berekent de rest van de deling van de bovenste 2 elementen.
MAX neemt de grootste van de bovenste 2 elementen.
MIN neemt de kleinste van de bovenste 2 elementen.
ABS berekent de absolute waarde van het bovenste getal.
MINUS draait het teken van het bovenste element om.
1+ telt 1 op bij het bovenste element.
2+ telt 2 op bij het bovenste element.
2* vermenigvuldigt het bovenste element met 2. Dit is sneller dan: 2 *

Dit waren enkele rekenkundige bewerkingen, maar er zijn ook andere bewerkingen op de stack mogelijk:

DUP dupliceert het bovenste element. 1 2 DUP geeft: 1 2 2
DROP verwijdert het bovenste element. 1 2 3 DROP geeft: 1 2
SWAP verwisselt de bovenste 2 elementen. 1 2 SWAP geeft: 2 1
OVER copieert het 2e element van de stack bovenop de stack. 6 4 3 2 OVER geeft: 6 4 3 2 3
ROT verwijdert het 3e element en plaatst het boven op de stack. 4 5 6 7 8 ROT geeft: 4 5 7 8 6
I PICK copieert het I-de element van de stack naar de top van de stack. 1 2 3 4 5 6 4 PICK geeft: 1 2 3 4 5 6 3
I ROLL verwijdert het I-de element en plaatst het boven op de stack. 1 2 3 4 5 6 4 ROLL geeft: 1 2 4 5 6 3

We zien nu dat:
1 PICK is equivalent aan DUP
2 PICK is equivalent aan OVER
2 ROLL is equivalent aan SWAP
3 ROLL is equivalent aan ROT

FORTH kent ook logische operatoren:

AND berekent de bit-voor-bit logische AND van de 2 bovenste elementen. 15 21 AND geeft: 5
OR berekent de bit-voor -bit logische OF van de 2 bovenste elementen. 15 21 OR geeft: 31
XOR berekent de bit-voor-bit logische exclusieve-OF van de bovenste 2 elementen. 15 21 XOR geeft: 26

INPUT EN OUTPUT IN FORTH:

FORTH kent de volgende INPUT/OUTPUT routines:

. haalt het bovenste getal van de stack en drukt het af.
EMIT haalt het bovenste getal van de stack en drukt het af, als ASCII character. 64 DUP EMIT . geeft:@ 64 OK
KEY leest een character in van het toetsenbord en zet het als getal op de stack.
." drukt een tekst af. De tekst moet worden afgesloten met " ." HALLO" geeft: HALLO
CR spring naar het begin van de volgende regel.

Inwendig rekent ACORN-FORTH altijd binair. De input en output kan echter geschieden in elk gewenste talstelsel. Standaard zijn aanwezig de 'decimale' en de 'hexa-decimale' basis. Omschakelen gebeurt via de commando's: "HEX" en "DECIMAL". Hoe je kan overgaan op een ander talstelsel komt nog.

HET PROGRAMMEREN IN FORTH:

Programmeren in FORTH is eigenlijk niets anders dan het toevoegen aan de standaard-FORTH functies van je eigen functies. Als je vindt dat een functie ontbreekt, dan maak je hem gewoon zelf! Deze zelf-gemaakte functies zijn dan op dezelfde manier te gebruiken als de standaard functies. We demonstreren dit weer aan de hand van een voorbeeld:

We typen in: : X↑2 DUP * ;

Wat hebben we nu eigenlijk gedaan:

:   geeft aan dat we een nieuwe functie gaan definieren.
X↑2   dit is de naam van onze nieuwe operator. Deze wordt altijd meteen
      na de : gezet.
DUP * dit is wat de nieuwe operator moet doen. In dit geval moet dus het
      kwadraat van het bovenste getal op de stack worden berekend.
;   hiermee wordt de definitie afgesloten.

Typen we nu vervolgens: 5 X↑2 .
dan zal de computer antwoorden met: 25 OK.
We zien dat de nieuwe functie op dezelfde manier gebruikt wordt als de standaard-FORTH functies. Hij kan dus ook in alle daarna volgende definities gebruikt worden. Bv.:   : X↑4 X↑2 X↑2 ;
3 X↑4 .   geeft nu: 81 OK
In ACORN-FORTH mag een naam van een functie max. 31 karakters lang zijn. Een naam kan nooit spaties bevatten.
Een FORTH programma bestaat uit een rij zelf-gemaakte functies, die steeds verder gecombineerd worden. Het gehele FORTH programma wordt uiteindelijk gerepresenteerd door een enkel woord. Om een programma in FORTH te schrijven, moet de taak van het programma in steeds kleinere taakjes worden verdeeld. Uiteindelijk blijven dan zeer kleine taken over, die makkelijk in FORTH kunnen worden geschreven. Deze worden dan gecombineerd tot een heel computer-programma. Wij zullen hiervan nog voorbeelden zien.
We zullen nog een operatie definieren: : - 3 * ;
We hebben nu de operatie "-" gedefinieerd als "vermenigvuldig met 3". We zien dus dat verschillende functies dezelfde naam mogen hebben. Alleen de laatste definitie van een functie kunnen we gebruiken, dus:
Als we intypen: 5 3 - .
dan zal de computer reageren met: 9 OK
We kunnen nu dus niet meer 2 getallen van elkaar aftrekken. Daarom is er een functie in FORTH, die zorgt dat we deze definitie ongedaan kunnen maken: FORGET X↑4   FORTH is nu "X↑4" en alle latere definities vergeten
De standaard-FORTH functies zijn beschermd.
FORGET DROP levert een foutmelding op.
Als we willen weten welke definities we ter beschikking hebben dan kunnen we gebruik maken van:
VLIST   we krijgen dan een lijst van alle zelf-gemaakte en alle standaard-FORTH definities.

DEEL 2

VARIABELEN IN FORTH:
Het is in FORTH ook mogelijk om variabelen te gebruiken. Ze worden op de volgende manier gedefinieerd: 44 VARIABLE HOEVEEL
De variabele met de naam "HOEVEEL" hebben we gedefinieerd en we hebben hem om te beginnen de waarde 44 gegeven. Als we de waarde van "HOEVEEL" willen gebruiken ineen berekening, dan moet deze waarde eerst op de stack worden gezet: HOEVEEL @
Op de top van de stack staat nu de waarde van "HOEVEEL" (=44). Het veranderen van de waarde van een variabele gaat op de volgende manier:
HOEVEEL !
Het bovenste getal is nu van de stack gehaald en aan de variabele "HOEVEEL" toegekend. We kennen nog een bewerking voor variabelen:
1 HOEVEEL +!
De waarde van "HOEVEEL" is nu met 1 opgehoogd.

DE "DO-LOOP" IN FORTH:
We zullen een voorbeeld laten zien: : TELLER 6 1 DO I . LOOP ;
Wat is er gebeurd?
:   dit geeft aan dat we een nieuwe functie definieren.
TELLER dit is de naam van de nieuwe functie.

6 1 de getallen 6 en 1 worden op de stack gezet.
DO dit is het begin van onze "DO-LOOP". De getallen 6 en 1 worden van de stack gehaald. De LOOP-index krijgt om te beginnen de waarde van het bovenste getal (=1). Zodra de LOOP-index de waarde van het 2e getal (=6) bereikt, wordt er gestopt.
I de waarde van de LOOP-index wordt op de stack gezet.
. het bovenste getal wordt van de stack gehaald en afgedrukt. In dit geval is dat dus steeds de LOOP-index.
LOOP dit markeert het einde van de "DO-LOOP". De index wordt met 1 verhoogd en we gaan terug naar "DO".
; einde van deze definitie.

Typen we nu in: TELLER
dan zal de computer reageren met: 1 2 3 4 5 OK

We moeten op de volgende dingen letten:
- Een DO-LOOP kan alleen binnen een definitie gebruikt worden.
- DO haalt 2 waarden van de stack. Hoe deze waarden daar gekomen zijn doet er niet toe. Ze mogen dus ook buiten de definitie, waarin de DO-LOOP voorkomt, op de stack worden gezet.
- DO-LOOP's mogen genest worden. De functie "I" zet altijd de waarde van de index van de binnenste DO-LOOP op de stack.

Nog een paar v.b.:

```
: TAFEL 11 1 DO DUP I * . LOOP ;
3 TAFEL     geeft: 3 6 9 12 15 18 21 24 27 30 OK
: WACHT 0 DO LOOP ;
8 WACHT           geeft: een korte pause
3000 WACHT        geeft: een lange pause
```

DE IF-ELSE-THEN CONSTRUCTIE:

Ook de IF-ELSE-THEN constructie kan alleen binnen een definitie voorkomt.
We bekijken een v.b.: `: GELIJK? = IF ." GELIJK " ELSE ." ONGELIJK " THEN ;`
Wat is er gebeurd?

: GELIJK? dit is de naam van de definitie.
= de 2 bovenste getallen worden van de stack gehaald. Als ze gelijk zijn, dan wordt er een getal op de stack gezet dat de logische waarde "WAAR" representeert, en als ze ongelijk zijn, dan wordt er een getal op de stack gezet met de logische waarde "NIET-WAAR".
IF er wordt getest of op de stack de logische waarde "WAAR" staat. Als dit zo is dan worden de operaties tussen "IF" en "ELSE" uitgevoerd. Staat op de stack de logische waarde "NIET-WAAR", dan worden de operaties tussen "ELSE" en "THEN" uitgevoerd.
." GELIJK " druk af "GELIJK".
ELSE onderdeel van IF-THEN-ELSE constructie.
." ONGELIJK " druk af "ONGELIJK".
THEN afsluiting van de IF-THEN-ELSE constructie. Na het uitvoeren van de operaties tussen "IF" en "ELSE" of tussen "ELSE" en "THEN", gaat het programma verder met de operaties na "THEN".
; afsluiting van de definitie.

```
3 5 GELIJK?   geeft:   ONGELIJK OK
7 7 GELIJK?   geeft:   GELIJK OK
```

Binnen FORTH wordt de logische waarde "NIET-WAAR" gerepresenteerd door het getal 0. Ieder getal ongelijk aan nul representeert de logische waarde "WAAR". De relationele operaties in FORTH, waartoe ook de operatie "=" behoort, zetten op de stack het getal 1 als de uitkomst moet zijn "WAAR".

```
Dus:   7 3 = .     geeft:    0 OK
       5 5 = .     geeft:    1 OK
```

FORTH kent de volgende relationele operaties:

0= zet een 1 op de stack als het bovenste getal gelijk is aan 0. Is het bovenste getal niet gelijk aan 0, dan komt er een 0 op de stack.

0< zet een 1 op de stack als het bovenste getal negatief is. Is het bovenste getal niet-negatief dan komt er een 0 op de stack.

= zet een 1 op de stack als de 2 bovenste getallen gelijk zijn. Zijn de 2 getallen ongelijk dan komt er een 0 op de stack.

< zet een 1 op de stack als het bovenste getal kleiner is dan het 2e getal op de stack. In alle andere gevallen komt er een 0 op de stack.

> zet een 1 op de stack als het bovenste getal groter is dan het 2e getal op de stack. In alle andere gevallen komt er een 0 op de stack.

```
5 5 < .        geeft:    0 OK
0 0= .         geeft:    1 OK
8 0< .         geeft:    0 OK
-153 0< .      geeft:    1 OK
```

BEGIN-UNTIL:

We bekijken een v.b.: : DRUK-AF-TOT-0 BEGIN DUP . 0= UNTIL ;

Verklaring:

```
: DRUK-AF-TOT-0  is het begin van de definitie.
BEGIN            dit is het begin van de BEGIN-UNTIL constructie.
DUP .            druk het bovenste getal van de stack af.
0=               kijk of dit getal gelijk is aan 0.
UNTIL            ga door totdat de voorgaande test het antwoord "WAAR"
                 oplevert. Als de voorgaande test "NIET-WAAR" oplevert
                 ga dan terug naar "BEGIN".
;                einde van de definitie.
1 2 3 0 4 7 3 1 DRUK-AF-TOT-0     geeft:    1 3 7 4 0 OK
```

Als illustratie kan het volgende voorbeeld dienen. Het berekent de grootste gemene deler van 2 getallen.

```
: COPY OVER OVER ;
: G-G-D BEGIN COPY - ABS ROT MIN DUP 0= UNTIL DROP ;
: GGD CR ." DE G.G.D VAN " COPY . ." EN " . ." IS " G-G-D . ;
9 15 GGD        geeft:    DE G.G.D VAN 15 EN 9 IS 3 OK
221 119 GGD     geeft:    DE G.G.D VAN 119 EN 221 IS 17 OK
```

De berekening van het kleinste gemene veelvoud gaat als volgt:

```
: KGV COPY * ROT ROT G-G-D / ;
8 12 KGV .      geeft:    24 OK
```

BEGIN-WHILE-REPEAT:

Ook nu beginnen we met een v.b.:

```
: DELETEO BEGIN DUP 0= WHILE DROP REPEAT ;
```

Verklaring:

```
:        begin van de definitie.
DELETEO  naam van deze functie.
BEGIN    dit is het begin van de BEGIN-WHILE-REPEAT constructie.
DUP 0=   we testen of het bovenste getal op de stack gelijk is aan 0.
WHILE    als de voorgaande test het resultaat "WAAR" heeft, dan wordt
         verder gegaan met de volgende operatie. Was het resultaat "ON-
         WAAR", dan springen we naar de operatie achter "REPEAT".
DROP     als het bovenste getal gelijk is aan 0, verwijder het dan.
REPEAT   spring terug naar "BEGIN".
;        einde van de definitie.
```

Met deze operatie kunnen we alle nullen boven op de stack verwijderen.

```
1 2 3 0 0 9 0 0 0 DELETEO          geeft:  9 0 0 3 2 OK
```

We kunnen nu een v.b. programmaatje laten zien. Het is een spelletje. De speler moet een getal tussen 1 en 1023 in gedachten nemen, de computer zal proberen het getal te raden.

```
: MIDDEL OVER OVER + 2 / ;
: VRAAG1 CR ."  IS HET " MIDDEL . ." ? " KEY DUP EMIT ;
: VRAAG2 CR ."  GROTER OF KLEINER? " KEY DUP EMIT ;
: RADEN 0 1024
        BEGIN VRAAG1 78 =
        WHILE VRAAG2 71 =
              IF MIDDEL ROT DROP SWAP
              ELSE MIDDEL SWAP DROP
              THEN
        REPEAT DROP DROP CR ."  GEVONDEN " CR ;
```

Het spel zou als volgt kunnen verlopen:

```
RADEN
IS HET 512? N
GROTER OF KLEINER? K
IS HET 256? J
GEVONDEN
OK
```

N.B. 78 is de ASCII waarde voor "N" en 71 is de ASCII waarde voor "G"!!!

FORTH opslaan. ***********

Jedereen, die in het bezit is van de 'Josbox' en de 16K CMOS uitbreidingskaart, kan op de volgende manier 'Atom-FORTH' opslaan in zijn pseudo-ROM geheugen.

```
*RUN"FORTH"
BREAK
COPY ≠240,≠3FF,≠6A40
COPY ≠2800,≠3BFF,≠6C00
?18=≠69
NEW
10 COPY ≠6C00,≠7FFF,≠2800
20 COPY ≠6A40,≠6BFF,≠240
30 LINK≠2800
40 END
```

Als men een volgende keer met FORTH wil werken, hoeft men niet het bandje te lezen! Opstarten van FORTH gaat als volgt:

```
?18=≠69
RUN
```

G.Akkermans.

LISP

INLEIDING:
De programmeertaal LISP is nu ook voor de ACORN ATOM beschikbaar. Het programma wordt op tape geleverd en is 5.5K groot (machinecode) met bijbehorend 3K data. De machinecode staat op positie ≠8200 tot ≠9800, en data op ≠2800 tot ≠3400. Voor de 12K ATOM past dit net (er is niet veel werkruimte meer over), maar met 16K uitbreiding en 1K 'piggy packing' werkt het prima.

ONTSTAAN VAN LISP:
Jaren gelden hield men zich al bezig met het (theoretische) probleem 'BEREKENBAARHEID'. Dit probleem houdt in: is er een formele definitie te vinden waarmee men voor een willekeurig probleem kan aantonen of het m.b.v. een rekenautomaat (bv. een ATOM) berekenbaar ofwel oplosbaar is. Er zijn de meest onbruikbare definities verzonnen zoals bv. van A.CHURCH: 'een probleem is effectief berekenbaar als het intuitief berekenbaar is'. Makkelijker gezegd volgens mij: 'als je denkt dat het kan, dan kan het ook'. Ook A.M.TURING heeft zich er mee bezig gehouden, en heeft de 'TURING MACHINE' verzonnen. Dit is een theoretisch, eenvoudige machine, waaraan lekker gerekend kan worden op papier, maar praktisch nut heeft hij niet. De wiskundige J.McCARTHY zat daar ook mee, en bedacht een andere theoretische ('elegante' zoals de wiskundigen zeggen) machine: LISP. Dit staat voor LISt Processing. Een paar studenten van McCARTHY zagen het praktisch nut van het ding in, en schreven er een interpretator voor op een IBM in 1960. Tot op de dag van vandaag is er nooit essentieel iets veranderd aan die eerste interpretator.

DE TAAL LISP:
LISP is een gewone programmeertaal net als alle andere talen zoals BASIC, met als enige verschil dat hij op het eerste gezicht totaal onleesbaar lijkt. Het eigenlijke verschil tussen LISP en andere talen is, dat bv. BASIC een 'ITERATIEVE' taal is, d.w.z. er worden opdrachten (statements) achter elkaar uitgevoerd en LISP is een 'FUNCTIONELE' taal, d.w.z. er worden functies toegepast op 'argumenten'. Iedereen die wat met wiskunde te maken heeft (gehad), heeft in feite 'gelispt'. B.v. zo'n vervelend sommetje: F(X)=X↑4-X↑2+1 en daar worden dan allerlei vragen over gesteld zoals: 'Wat is F(7)?' of iets dergelijks. Als je dat dan zat uit te rekenen zat je in feite voor LISP-INTERPRETATOR te spelen, want het enige wat LISP kan is FUNCTIES uitrekenen (of netter gezegd: evalueren). Deze functies bestaan uit een naam (hier 'F') met een (of meerdere) argumenten of formele parameters (hier 'X') gevolgd door een definitie van de functie (hier 'X↑4-X↑2+1'). De functie 'F' is beschreven m.b.v. andere functies '↑','-' en '+'. Het inconsequente van de mens is dan weer, dat we dan niet opschrijven '↑X4' zoals wel zou moeten volgens de definitie van een functie (eerst de FUNCTIENAAM, dan de ARGUMENTEN). LISP is wel zo consequent. In LISP staat altijd de naam van de functie vooraan gevolgd door z'n argumenten, het geheel staat tussen haakjes. Bovenstaande functie zou er in LISP uitzien als: (+(-(↑ X 4)(↑ X 2))1) en het geheel heeft de naam 'F'.

Als LISP nu het bovenstaande uit zou moeten rekenen voor een X-waarde 7, dan doet hij dat als volgt:
1) tel op '+' (-(↑ X 4)(↑ X 2)) en 1
dit houdt in, dat er uigerekend moet worden:
2) trek af '-' (↑ X 4) en (↑ X 2)
dit houdt weer in dat er uitgerekend moet worden:
3) verhef '↑' X en 4
4) verhef '↑' X en 2
Er kan dus pas wat worden uitgerekend binnen haakjes, als alles wat tus-

sen nog meer haakjes staat al is uitgerekend.
LISP wordt LISt Processing genoemd, omdat alles wat LISP kent: ('FUNCTIENAAM' 'ARGUMENTØ' . . . . 'ARGUMENTn') op te vatten is als een lijst met n+2 elementen. LISP interpreteert elke lijst als volgt: Het eerste element van de lijst is de functie-naam en alles wat daar achter komt zijn de argumenten van die functie. Is een willekeurig argument ook weer een lijst (haakjes!!), dan wordt deze weer hetzelfde bekeken (functie plus argumenten). Is een bep. argument niet iets tussen haakjes, dan wordt hiervan de "waarde genomen" (in het voorbeeld X heeft de waarde 7). De waarde van zoiets kan bij LISP een getal zijn of weer een hele lijst met allemaal haakjes.
Formeler gezegd: LISP kent:
1) ATOMEN dit zijn of getallen of strings beginnend met een letter en eventueel gevolgd door letters en/of cijfers.
2) LIJSTEN dit zijn lijsten van lijsten en/of ATOMEN.
Tesamen vormen deze 2 de zg. 'symbolische expressies'. Een SEXPR (afkorting) is dus of een lijst of een atoom. Een voorbeeld van een SEXPR is de bovenbeschreven functie.
LISP heeft een aantal standaard functies tot z'n beschikking staan, waarmee nieuwe functies gedefinieerd kunnen worden. Dit is een van de sterke kanten van LISP. Vergelijk bv. de 'DEF FN' van sommige BASIC's en de 'FUNCTION' van PASCAL. Programma's in LISP geschreven, zijn over het algemeen veel korter dan programma's in andere talen geschreven. LISP programma's zijn echter meestal langer als het gaat om numerieke toepassingen waarin veel gerekend moet worden met eenvoudige datastructuren zoals ARRAYS enz. LISP kent maar 1 datastructuur en dat is de symbolische expressie (of de LIJST). Het leuke is nu, dat een LISP programma zelf OOK die datastructuur bezit. Een programma kan dus DATA zijn voor een (eventueel hetzelfde) programma, terwijl DATA een programma kan zijn. Met LISP programma's kunnen op een gemakkelijke manier LISP programma's geschreven worden. Probeer dat eens met een taal als ALGOL, PASCAL of met de meeste BASIC soorten (ATOM BASIC is wat dat aangaat een beetje slimmer dan de anderen). Een ander sterk punt van LISP is, dat 'RECURSIE' geen enkel probleem oplevert. Even wat vooruit lopen, een voorbeeldje.
N! (N FACULTEIT is in ATOM BASIC als volgt te berekenen:

```
10 INPUT"N="N
20 R=1
30 IF N=0 GOTO 70
40 FOR H=1 TO N
50 R=R*H
60 NEXT H
70 PRINT"N!"R
80 END
```

Toch een heel gedoe met een loop en wat testwerk. Met LISP gaat dat heel anders.
De definitie van N! is: Q! = 1
N! = N*(N-1)!
Dus in LISP schrijf je op:

```
(COND ((EQ N 0) 1)
      (T(TIMES N (FAC(DIFFERENCE N 1)))))
```

Het geheel heet "FAC". De functie : "COND" is de "IF" van BASIC
" " : "TIMES" vermenigvuldigt
" " : "EQ" test op gelijkheid
" " : "T" is "TRUE"
" " : "DIFFERENCE" is de "-" van BASIC

Nog even herhalen, een LISP programma is een lijst. Een lijst is: a) Een eventueel leeg woord of b) Een lijst van lijsten. Als een lijst niet een enkel woord is, dan staan de elementen van die lijst tussen haakjes. Bv. (EEN TWEE DRIE) of (VIER ((VIJF ZES)) enz.
Het eerste element van een lijst is de FUNCTIENAAM en de eventuele andere elementen zijn de argumenten van de functie die bij die naam behoren. Alle LISP INTERPRETATOREN beschikken over een redelijk groot aantal standaard functies. M.b.v. deze standaard functies kan de gebruiker zijn eigen

functies samenstellen en zo een LISP programma schrijven. Dit klinkt misschien ingewikkelt of omslachtig, maar in BASIC gebeurt het precies hetzelfde, alleen wordt er dan niet gepraat over functies, maar over statements. Zo wordt bv. A=123 in BASIC een 'assignment' statement of 'toekennings opdracht' genoemd, maar met hetzelfde recht kan je praten in LISP over een functie bv. 'GEEFWAARDE' die 2 argumenten heeft. Hier zijn dat 'A' en '123'. De waarde van de functie is niet van belang.
De eerste LISP INTERPRETATOREN beschikten maar over een klein aantal standaard (of 'primitieve') functies. Dit is echter geen nadeel, daar alle functies die men maar kan verzinnen uit deze standaard functies samengesteld kunnen worden. J.McCarthy heeft m.b.v. LISP ook bewezen, dat alle 'effectief berekenbare' functies niets anders waren dan een samenstelsel van een klein aantal simpele functies.

De belangrijkste standaard functies van LISP zijn:
--- QUOTE ---
Dit is een simpele functie met een (1) argument, de waarde van de functie IS het argument. Toets maar eens in:
  (QUOTE HALLO)   het antwoord is:
  HALLO   Dit lijkt misschien eigenaardig of nutteloos, maar bedenk, dat LISP altijd van woorden de WAARDE probeert uit te rekenen ('evalueren'). Als je in had getoetst:
  HALLO   dan had LISP de WAARDE van HALLO geprobeerd te vinden, en dat was 'm waarschijnlijk niet zo best gelukt. Misschien verduidelijkt het volgende voorbeeld iets. Als je in BASIC intoetst:
  A=X   dan krijgt A niet de waarde 'X' (de letter X), maar de WAARDE van de letter X (of variabele), en dat is bij BASIC altijd een getal. Wil je in BASIC iets LETTERLIJK hebben, dan zet je het tussen QUOTES:
  $A="HALLO"   Nu heeft A de waarde HALLO en niet de waarde van HALLO, wat dat dan ook zijn mag.
--- CAR & CDR ---
De betekenis van de namen van deze functies is niet zo gemakkelijk te achterhalen zoals bv. met 'PLUS' of 'REMAINDER'. De namen stammen uit de grijze oudheid van de IBM 7400 computer, waar de eerste LISP INTERPRETATOREN op gemaakt zijn. Deze computers hadden 30 BITS woordlengte, waarvan de eerste 15 BITS het ADRES gedeelte en de laatste 15 BITS het DATA gedeelte voorstelden van machinecode instructies. Vandaar de namen Contents of Adress part of Register en Contents of Data part of Register. Maar nu genoeg geschiedenis. Deze 2 functies hebben allebei een (1) argument, en dat moet een lijst zijn. Deze lijst mag niet leeg zijn en niet gelijk zijn aan een enkel woord. De waarde van de functie CAR is gelijk aan het eerste element van die lijst en de waarde van CDR is gelijk aan de lijst uitgezonderd het eerste element. Bv.
  (CAR (QUOTE (DIT IS EEN LIJST)))   heeft als waarde DIT. Let op het gebruik van QUOTE!!. Het argument van CAR is:
  (QUOTE (DIT IS EEN LIJST))   en LISP gaat dat evalueren. Het resultaat is:
  (DIT IS EEN LIJST)   (zie de functie QUOTE) en hiervan wordt het eerste element genomen dus: DIT. Hadden we ingetypt:
  (CAR (DIT IS EEN LIJST))   dan ziet LISP het woordje 'DIT' weer als een functie met als argumenten IS,EEN en LIJST, en dan was alles in de war gelopen waarschijnlijk. Bij CDR gebeurt er dit:
  (CDR (QUOTE (DIT IS EEN LIJST)))   De waarde is nu:
  (IS EEN LIJST)   De functies mogen ook door elkaar gebruikt worden:
  (CAR (CDR (QUOTE (DIT IS EEN LIJST)))) Dit heeft als waarde: IS.

Om veel typwerk te vermijden zijn in ATOMLISP de functies CAR,CDR,CAAR, CADR,CDAR,CDDR opgenomen. Deze laatste 4 functies vervangen:
(CAR (CAR .... (CAR (CDR .... (CDR (CAR .... (CDR (CDR ....
CAR en CDR splitsen een lijst in tweeen, de volgende functie knoopt ze weer aan elkaar.

--- CONS ---

Dit is een functie met 2 argumenten. De waarde van CONStruct is een lijst gelijk aan het tweede argument met ervoor het eerste argument geplakt. Bv.

```
( CONS ( QUOTE DIT ) ( QUOTE ( IS EEN LIJST )))    heeft als waarde:
( DIT IS EEN LIJST )
```

In ATOMLISP mag je om nog meer typwerk te vermijden i.p.v.: (QUOTE ZOMAARIETS) ook wel ' ZOMAARIETS typen, dat scheelt ook weer wat haakjes. Stel dat LIJST een normale niet lege lijst is en ook niet een enkel woord, dan geldt er voor CAR, CDR en CONS dat:

```
( CONS ( CAR LIJST ) ( CDR LIJST )) = LIJST
```

Let op bij CONS, dat het antwoord bij:

```
( CONS '( DIT IS ) '( EEN LIJST ))    niet
( DIT IS EEN LIJST )                  maar
(( DIT IS ) EEN LIJST )
```

is!! Het eerste argument wordt TOEGEVOEGD als nieuw eerste element van de tweede lijst. Nog een voorbeeldje:

```
(CONS(CADR LIJST)(CONS(CAR LIJST)(CDDR LIJST)))    heeft als waarde:
( IS DIT EEN LIJST )
```

Het woord 'LIJST' heeft de waarde: (DIT IS EEN LIJST). Dit laatste voorbeeldje was al bijna niet meer te lezen door de haakjes. Er is dan ook een nette manier gevonden om LISP programma's leesbaar op te schrijven in de volgende vorm:

```
( FUNCTIENAAM ARGUMENT1
              ARGUMENT2
                  .
                  .
              ARGUMENTn)
```

Dus het laatste voorbeeld had er dan zo uitgezien:

```
( CONS ( CADR LIJST )
       ( CONS ( CAR LIJST )
              ( CDDR LIJST )))
```

Dit is al heel wat leesbaarder. In ATOMLISP is een functie opgenomen, een zg. 'PRETTY PRINTER' of 'SUPER PRINTER' die dat ook doet. Z'n naam is SPRINT. Dat vergemakkelijkt alles een beetje en vermijdt een hoop getel van al die haakjes.

Deze keer de predicaten van LISP. Predicaten zijn functies net als alle andere functies, met als enige verschil dat de functie maar 2 waarden kan hebben: WAAR of ONWAAR. In LISP wordt dit voorgesteld door 'T' voor waar en 'NIL' voor onwaar. De meest gebruikte predicaten (of testfuncties) zijn:

ATOM: deze functie is T als z'n argument een atoom is, anders NIL
LISTP: deze functie is het omgekeerde van ATOM
NULL: test of Z'n argument NIL is, zo ja dan T anders NIL
NOT: is gelijk aan NULL
EQ: test of z'n beide argumenten atomen zijn en gelijk zijn

Een paar predicaten speciaal voor getallen:

GREATERP: test of z'n eerste argument groter is als z'n tweede
LESSP: het omgekeerde van GREATERP
MINUSP: test of het argument negatief is of niet
ZEROP: test of z'n argument NUL (niet te verwarren met NIL) is

De hoofdletter P aan het eind van de predicatennamen staat voor 'predicaat'. Een paar voorbeeldjes:

```
(GREATERP 3 2) = T      (LESSP 3 2) = NIL      (ATOM 'A)     = T
```

(NOT(LISTP 'A) = T   (NULL(NULL(NULL NIL))) = T   (EQ 3 3) = T
(ZEROP 0) = T   (EQ 'A 'B) = NIL   (EQ 'A 'A) = T

--- COND ---

Een van de werkpaarden van LISP is de functie COND ('condition' te vergelijken met IF THEN ELSE). De structuur van de functie COND is even wennen:

```
(COND ((TEST1)(WAARDE1))
      ((TEST2)(WAARDE2))
         .       .
         .       .
      ((TESTn)(WAARDEn))
```

Dit kan vertaald worden als volgt:
IF TEST1 = WAAR THEN FUNCTIEWAARDE = WAARDE1 ELSE
IF TEST2 = WAAR THEN FUNCTIEWAARDE = WAARDE2 ELSE ... etc.

Als geen enkele test waar is, dan is de functiewaarde NIL. Een voorbeeld: Stel we willen het max. bepalen van 2 getallen X en Y.

```
(COND((LESSP X Y)Y)
     (T X))
```

--- DEFUN ---

Een ander veel gebruikte functie is DEFUN (functie definitie). De structuur is als volgt:

```
(DEFUN FUNCTIENAAM ( ARGUMENT1 ... ARGUMENTn) FUNCTIE )
```

Willen we bv. de functie MAXIMUM definieren, dan gaat dat als volgt:

```
(DEFUN MAXIMUM(X Y)
   (COND((LESSP X Y)Y)
        (T X)))
```

We kunnen deze functie gewoon gebruiken zoals andere functies, bv. (MAXIMUM 7 5).

--- SETQ & SET ---

Om een waarde aan iets toe te kennen kent LISP 2 functies: SETQ en SET. Voor de duidelijkheid even terug naar BASIC. Als we intypen A=B, dan wordt de waarde van A gelijk aan de waarde van B. Typen we in $A="B" dan wordt de waarde van A gelijk gesteld aan B. De structuur van SETQ en SET:

```
(SET[Q] ARGUMENT1 ARGUMENT2)
```

SETQ geeft dan de waarde van ARGUMENT2 aan ARGUMENT1 en SET geeft de waarde van ARGUMENT2 aan de WAARDE(!!!) van ARGUMENT1. Laten we er mee spelen:

(SETQ A 'B)   A heeft nu de waarde B
(SET A 'C)   De waarde van A (=B) heeft nu de waarde C ofwel B heeft de waarde C.
(SETQ A B)   A heeft nu de waarde die B ook had ofwel C.

De letter Q aan het eind van SETQ staat voor QUOTE. In LISP's spraakgebruik heet het, dat SETQ zijn eerste argument automatisch QUOTE (werkwoord) en dus niet evalueert.

Laten we eens een wat groter voorbeeld bekijken, en een beetje met de verzamelingenleer gaan rommelen. Als conventie spreken we af, dat in LISP een verzameling een lijst is en een element van een verzameling een atoom is. Dus (A B C D E) is een verzameling en A en B zijn elementen uit deze verzameling. () is ook een verzameling n.l. de lege verzameling. Laten we eens een functie definieren die bepaalt of een atoom een element is van een bep. verzameling.

```
(DEFUN MEMBER(X Y)
   (COND((NULL Y)NIL)
        ((EQ X(CAR Y))T)
        (T(MEMBER X(CDR Y)))))
```

Je kan de functie als volgt lezen: X is een atoom en Y is een verzameling. Is Y de lege verzameling, dan kan X daar geen element van zijn. Anders als X het eerste element is van Y dan T. Anders kijken we of X een element is van de rest van de verzameling Y ('t was niet het eerste element).

Als we een element toe willen voegen aan een verzameling, dan moeten we dus eerst kijken of dat element er al in zit, zo ja dan doen we niets, anders voegen we het element toe (dubbele postzegels beschouwen we niet als een element van de postzegel verzameling.

```
(DEFUN INSET(X Y)
   (COND((MEMBER X Y)Y)
        (T(CONS X Y))))
```

Als we een element willen verwijderen kijken we eerst of er wel iets te verwijderen valt. Zo ja, dan kijken we of het element dat we er uit willen halen er wel inzit. Zo ja dan halen we het er uit anders doen we niets.

```
(DEFUN OUTSET(X Y)
   (CIND((NULL Y)NIL)
        ((EQ X(CAR Y))(CDR Y))
        (T(CONS(CAR Y)(OUTSET X(CDR Y))))))
```

We kunnen nu werken met EEN verzameling en EEN element. Nu met 2 verzamelingen. Willen we de vereniging van 2 verzamelingen weten, ofwel het hele handeltje op een hoopje en de dubbler er uit halen, dan kan dat zoals hier. De truuk is als volgt: we halen steeds een element uit X en voegen dat toe aan Y totdat X leeg is.

```
(DEFUN UNION(X Y)
   (COND((NULL X)Y)
        (T(UNION(CDR X)(INSET(CAR X)Y)))))
```

Als we alle elementen van een verzameling Y willen hebben die niet in een verzameling X zitten kan dat zoals hiernaast.

```
(DEFUN DIFF(X Y)
   (COND((NULL X)Y)
        (T(DIFF(CDR X)(OUTSET(CAR X)Y)))))
```

En alle elementen die OF in X OF in Y zitten krijgen we als volgt te pakken.

```
(DEFUN SYMDIFF(X Y)
   (UNION(DIFF X Y)
         (DIFF Y X)))
```

Alle elementen die in beide verzamelingen zitten zijn als volgt op te sporen:

```
(DEFUN INTER(X Y)
   (SYMDIFF(UNION X Y)
           (SYMDIFF X Y)))
```

Wat is nu het praktisch nut van al deze functies? We kunnen het begin makan van een DATA BASE. In de praktijk zal niemand een DATA BASE in LISP gaan schrijven, maar als prototype voldoet LISP uitstekend. Werkt het eenmaal in LISP dan kan het hele handeltje omgeschreven worden naar een andere snellere taal. Deze manier van software ontwikkeling bij grote projekten is in de praktijk de meest gebruikelijke. Eerst uittesten in een logische theoretische taal en werkt dat eindelijk, dan omschrijven naar een of andere rare productietaal zoals PL1 of RPG2 of COBOL noem maar op.

Laten we een beetje gaan rommelen met bovenbeschreven functies. Stel we hebben drie kleine clubs: CLUB1, CLUB2 en CLUB3.

```
(SETQ CLUB1 '(JAN PIET KLAAS MARIE))
(SETQ CLUB2 '(KEES ANDRE MARION NELLIE))
(SETQ CLUB3 '(JOOP GERARD))
```

Jaap meldt zich aan als nieuw lid bij CLUB3:

```
(SETQ CLUB3 (INSET 'JAAP CLUB3))
```

Joop was vergeten dat hij al lid was:

```
(SETQ CLUB3 (INSET 'JOOP CLUB3))
```

Joop wil ook lid worden van CLUB2 en Kees ook van CLUB1 en CLUB3:

```
(SETQ CLUB2 (INSET 'JOOP CLUB2))
(SETQ CLUB1 (INSET 'KEES CLUB1))
(SETQ CLUB3 (INSET 'KEES CLUB3))
```

Nellie heeft er geen zin meer in:

```
(SETQ CLUB2 (OUTSET 'NELLIE CLUB2))
```

En ga zo maar door. Vragen als: wie zit er in CLUB1 of CLUB3 maar niet in CLUB2; CLUB1 en CLUB2 fuseren maar de mensen die ook in CLUB3 zitten hebben daar geen zin in en gaan er uit ...etc. zijn gemakkelijk te beantwoorden en administratief bij te werken m.b.v. dit kleine aantal LISP functies.

Nog een paar voorbeeldjes om de RECURSIE van LISP te laten zien.

Hoe plak je 2 lijsten aan elkaar? (CONS LIJST1 LIJST2) werkt niet, kijk maar:

```
(SETQ LIJST1 '(A B C))
(SETQ LIJST2 '(C D E))
(CONS LIJST1 LIJST2) = ((A B C)C D E)
```

De eerste lijst wordt het eerste element van lijst twee. Zo gaat het wel:

```
(DEFUN PLAKAAN(X Y)
   (COND((NULL X)Y)
        (T(CONS(CAR X)
              (PLAKAAN(CDR X)Y)))))
```

Hoe draai je een lijst om:

```
(DEFUN DRAAIOM(X)
   (COND((NULL X)X)
        (T(PLAKAAN(DRAAIOM(CDR X))
                (CONS(CAR X)NIL)))))
```

Hoe bepaal je of twee lijsten gelijk zijn:

```
(DEFUN GELIJK(X Y)
   (COND((EQ X Y)T)
        ((ATOM X)NIL)
        ((ATOM Y)NIL)
        ((GELIJK(CAR X)(CAR Y))(GELIJK(CDR X)(CDR Y)))
        (T NIL)))
```

Hoe zie je of een lijst achterstevoren gelijk is aan de lijst zelf:

```
(DEFUN SPIEGEL(X)
   (GELIJK X)(DRAAIOM X)))
```

Hoe krijg je het laatste element van een lijst te pakken:

```
(DEFUN LAATSTE(X)
   (CAR(DRAAIOM X)))
```

Om te weten te komen welke functies ATOM LISP kent kan je intypen:(OBLIST). Dit is een functie die als waarde een lijst geeft met als elementen de namen van alle (eventueel door de gebruiker gedefinieerde) functies. OBLIST is een afkorting van DEFINED OBJECTS LIST.

In het rekenwerk is LISP niet zo sterk, de volgende functies zijn standaard gedefinieerd:

```
(PLUS X Y)       = X+Y
(DIFFERENCE X Y) = X-Y
(TIMES X Y)      = X*Y
(QUOTIENT X Y)   = X/Y
(REMAINDER X Y)  = X%Y
```

In de eerste LISP INTERPRETATOREN waren alleen PLUS en DIFFERENCE gedefinieerd, de rest werd m.b.v. deze twee gedefinieerd:

```
(DEFUN TIMES(X Y)
   (COND((ZEROP X)Y)
        (T(PLUS X(TIMES(DIFFERENCE X 1)Y)))))
(DEFUN QUOTIENT(X Y)
   (COND((LESSP X Y)0)
        (T(PLUS 1(QUOTIENT(DIFFERENCE X Y)Y)))))
(DEFUN REMAINDER(X Y)
   (DIFFERENCE X(TIMES Y(QOUTIENT X Y))))
```

Ik hoop dat deze chaotische en/of moeilijke LISP aflevering een beetje duidelijk heeft kunnen maken wat LISP is en wat de programmeer discipline is die LISP met zich mee brengt. Vele mensen vinden LISP of afschuwelijk of zijn net als ik een LISP freak. Succes ermee.

Jos

; PILOT

WAT IS PILOT?

PILOT is een programmeertaal voor educatieve doeleinden. Deze taal leent zich uitstekend voor het maken van vraag en antwoord spelletjes, ofwel voor 'Computer Aided Instruction'. Het is een zeer simpele taal. ATOM-PILOT kent slechts 12 functies. De bijbehorende editor kent slechts 7 functies. Het totaal neemt nog geen 1Kbyte aan geheugenruimte in, zodat het zelfs op een ACORN ATOM met totaal 3 Kbyte RAM-geheugen kan werken. Natuurlijk kan men van zo'n kleine editor en interpreter geen wonderen verwachten.

DE EDITOR:

De editor van de ATOM-PILOT kent de volgende functies:

[illegible] Voer het programma uit.
Zet de editor-wijzer op de start van het programma.

\ Laat de volgende regel van het programma zien.

& Vul de regel aan met spaties, tot aan het eerstvolgende 'EINDE VAN DE REGEL'-karakter. De editor kent geen insert functie. Wel kan een oude regel, door een nieuwe regel worden overschreven, mits deze maar korter is dan de oude regel. (Evenlang kan ook, maar dan heb je deze functie niet nodig.) De nieuwe regel moet dan aan het einde aangevuld worden met spaties.

DELETE Verwijder het zojuist ingetypte karakter.

RETURN Geef aan het 'EINDE VAN DE REGEL'.

ESC Spring uit de PILOT editor naar de ACORN ATOM monitor. Het kommando laat het laatste adres van het PILOT programma zien. Dit adres heb je nodig als je het programma wilt opnemen op cassette. Dat kan via *SAVE.

Ieder ander ingetypt karakter wordt in de programma-tekst opgenomen.

DE PILOT-FUNCTIES:

ATOM-PILOT kent de volgende functies:

T:TEKST Druk af 'TEKST'.

A: Wacht op invoer. De input-string wordt bewaard in het antwoordveld. Het adres van deze opdracht wordt bewaard voor de 'J:' opdracht.

?: Wacht op invoer. De input-string wordt bewaard in het antwoordveld en in het naam-veld.

M:TEKST Vergelijk 'TEKST' met de inhoud van het antwoord-veld. Indien ze gelijk zijn, zet dan de MATCH-vlag op 'Y', anders wordt de MATCH-vlag 'N'. Deze functie accepteert meerdere argumenten, mits deze gescheiden zijn door een komma. Indien een van de argumenten het antwoord 'Y' oplevert, dan is het totaal resultaat ook 'Y'.

J:N Spring naar label 'N'. J:A betekent spring naar de laatste 'A:' opdracht en J:* betekent spring naar het begin van het programma.

U:N Spring naar subroutine 'N'. Het adres van de volgende regel wordt bewaard.

E: Return van subroutine.

S: Stop de uitvoering van het programma en ga terug naar de PILOT editor.

R:TEKST 'TEKST is commentaar bij het programma.

H: Maak het scherm schoon.

B: Geef een pieptoontje.

C:EXPR. Bereken de volgende expressie. De variabelen zijn A-Z. De operaties zijn + - en =. Het bereik van de berekeningen is -999 tot +999. Er wordt decimaal gerekend. C:$= zal het resultaat in het antwoordveld zetten.

We zien dat alle opdrachten bestaan uit een karakter gevolgd door een ':'. De letters zijn de eerste letters van de Engelse namen voor de functies.

T: TYPE	H: HOME AND CLEAR	E: EXIT
M: MATCH	R: REMARK	C: COMPUTE
U: USE SUBROUTINE	A: ACCEPT	B: BELL
S: STOP	J: JUMP	

Iedere opdracht mag worden voorafgegaan door een test . Deze test onderzoekt of de MATCH-vlag 'Y' of 'N' is. Wordt aan de test voldaan, dan wordt de opdracht uitgevoerd. Wordt er niet aan voldaan, dan slaan we de regel over. De test wordt aangegeven door de letter 'N' of 'Y'. Bijvoorbeeld:

```
  T:JA OF NEE?
  A:
  M:J,Y,OK,NAT
  YT:JA DUS.
  NT:NEE DUS.
  S:
```

Als bij A: geantwoord wordt met 'J' (of 'JA' of 'Y' of 'YES' of 'OK' of 'NATUURLIJK') dan wordt afgedrukt: JA DUS. Wordt er negatief geantwoord, dan verschijnt er : NEE DUS. Iedere opdracht of iedere test kan bovendien weer voorafgegaan worden door een label. Als label kunnen de letters B t/m Z gekozen worden. Een label wordt aangegeven door '*':

*XYT:WAAR betekent: label 'X'; als laatste MATCH 'Y' opleverde, druk dan af 'WAAR'.

VARIABELEN:

In de compute opdracht, wordt een variable voorgesteld door een letter. In een type of een match opdracht moet een variabele aangegeven worden door een '$'. Dus:

T:$K Betekent:druk af de waarde van variabele K.

Met '$?' wordt het naam-veld bedoeld. Zie bv.

```
  T:UW NAAM IS $?
```

Komt in het match statement een variable voor, dan worden de niet significante nullen onderdrukt. Bekijk het volgende eens:

```
  C:X=7
  T:TYP EEN GETAL.
  A:
  M:$X
  YT:GETAL = $X
  S:
```

De tekst 'GETAL = 7' wordt afgedrukt als we bij A: het volgende antwoorden: 7
70 t/m 79
700 t/m 799
Dit lijkt misschien vreemd, maar het is volledig in overeenstemming met de manier waarop het match-statement werkt op strings. Een voorbeeld:

```
  A:
  M:HA
  YT:Y
```

Levert 'Y' op bij alle invoer, die begint met 'HA' (bv.'HALLO'). Het match-statement is, mits verstandig gebruikt, een zeer krachtig stuk gereedschap.

In ATOM-PILOT is het niet mogelijk om direkt een numerieke waarde in te lezen in een variabele. Via een truck is dit probleem toch op te lossen. Een programmaatje, dat zo iets doet, staat op het bandje ('PILOT.VB1').

WERKEN MET ATOM-PILOT:

ATOM-PILOT wordt van de cassette geladen via *RUN of via *LOAD en LINK ≠2A00. De programmatekst begint op adres ≠2E00. Na 'ESC' wordt het eindadres van de programmatekst afgedrukt. Met *SAVE 2E00 XXXX kan het programma opgenomen worden. Een volgende keer kan het programma geladen worden met *LOAD en gestart met: LINK≠2A00 (ret)

Het einde van de programmatekst wordt opgezocht door te zoeken naar de eerste regel, waarin geen':' voorkomt bij de eerste 5 karakters. Om ervoor te zorgen, dat de zoekprocedure niet toevallig wel een ':' tegenkomt na het eind van het programma, kan simpelweg als laatste regel opgenomen worden: 'EINDE'

Probeert u het volgende maar eens:

```
*RUN"PILOT"
T:VEEL SUCCES!
P:
S:
EINDE
?
ESC
```

U kunt verder zelf spelenderwijs met deze taal vertrouwd raken. Heeft u nog vragen? Voor informatie kunt u bellen: 010-182253.

Gerard Akkermans.

Word Processor. "WORD PROC." ********
F.Carver stuurde een word-processor in, Berry Lam maakte nog een aangepaste versie ervan. Beide listings kunt u op het bandje vinden.

Editor. ******
Een editortje; gesuffeld en verbeterd door E.Wolters.
-88 line text processor- Users Group
This program allows the user to edit & print lines of text, up to 64 characters per line. All the usual keys are available for editing: COPY,REPT,DELETE etc. Return is pressed when the line is as required. An inverse R shows the position of the return on the line being edited, which is as far as the printer will print out. If there is no change to the line to be edited, just press return. Graphics memory is used to store text, 88 lines being available in an Atom with graphics mode 4. Adjust the 88 in lines 10, 120, 300, 305, 500 accordingly for smaller Atoms: Mode 3 - 40 lines, mode 2 - 16 lines, mode 1 - 8 lines.
Text can also be saved on tape and loaded at a later date. For teleprinter users, with the machine code for the print routine, in ≠2800 to ≠28FF: Add lines: 15 ?35=0;?36=≠3A 25 !≠208=≠FF94283E;P.$3
*SAVE the print routine and text processor together with:
*SAVE"88LTP" 2800 2C7E CE86 Use with *RUN"88LTP"
If the program is used without a printer, it will hang up because of line 310. To finisch editing enter 99 as the line number. This program is not suitable for unexpanded Atoms which use ≠8200 onwards for program text.

```
 10REM 88 LINE TEXT PROCESSOR
 20DIM A(70)
 25g?=0;GOS.u
 30P."     E - EDIT"'"     P - PRINT"'
   "     L - LOAD"'
 31IN."     S - SAVE"'"     F - FINISH"
   '''"YOUR CHOICE"$A
 40IF?A=CH"E";G.100
 50IF?A=CH"P";G.300
 60IF?A=CH"L";G.400
 70IF?A=CH"S";G.500
 80IF?A=CH"F";?≠E1=≠80;E.
 90G.g
100IN.'"ENTER LINE TO BE EDITED"'L;
   ?≠E1=≠80
110IFL=99;G.g
120IFL<1 OR L>88;G.100
130P=≠8200+(L-1)*≠40
140F.J=P TO (P+≠3F)
150IF?J=13;P."r";G.180
160IF?J<32 OR ?J>95;P." ";G.180
170P.$?J
180N.J
190P.$11$11$13$8
200IN.$A
210IF$A="";G.100
220F.J=0TO LEN(A)
230?P=A?J;P=P+1
240N.J;G.100
300IN.'"ENTER LAST LINE NR.(1-88)"L
305IF L<1 OR L>88;G.300
310P.$12
320F.K=1TO L
330P.=≠8200+(K-1)*≠40
340F.J=P TO P+≠3F
350IF?J=13;P.';J=≠FFFF;G.380
360IF?J<32OR?J>95;P." ";G.380
370P.$?J
380N.J
390N.K;P.$3;G.30
400IN."FILENAME "$A;?≠E1=≠80
410X=≠A0;!X=A;X!2=≠8200
420LINK≠FFE0;G.g
500IN.'"LINES 1-"L;?≠E1=≠80
510IFL<1 OR L>88;G.500
520L=≠8200+L*≠40
530IN."FILENAME "$A
540X=≠A0
550!X=A;X!2=≠8200;X!6=≠8200
560X?8=L;X?9=L/256
570LINK≠FFDD;G.g
720uP.$12;?≠E1=0
730F.W=1TO31;P.$≠A3;N.;P.';
   DOP.$≠DF;U.C.=11;P."editor"
740DOP.$≠DF;U.C.=31;P.';F.W=1TO31;
   P.$≠D0;N.;P.';R.
```

Bubble sort. "BUBBLE SORT" ********

Dit programma werd door H.Marks ingestuurd. Toelichting: Het grote voordeel van dit programma is, dat het niet strings zelf verschuift, maar de pointers naar de string verzet. Deze pointers staan in het array BBI en de strings staan vanaf ≠8200, zo compact mogelijk. 'Sorting level' is het aantal characters vanaf het begin van de string dat wordt bekenen. Dat levert dus een soort nauwkeurigheid van sorteren op. Als tijdens het invoeren van de strings blijkt dat het aantal te sorteren elementen te hoog is opgegeven voldoet het de string 'sort' in te voeren om de invoer af te breken.

Batcher's sort. "BATCHER" *********

Nadat Marks Bubble sort had ingezonden, ging hij zich nogmaals bezighouden met sorteren van strings in Basic. Dit had tot gevolg dat hij dusdanige vorderingen maakte dat Bubble-sort (in zijn ogen) hopeloos inefficient is geworden. Voordeel van dit nieuwe programma is de verhoogde snelheid, de tijd per string wordt korter bij hogere aantallen strings. Dat wordt veroorzaakt doordat er relatief veel tijd in wat initialisaties, in het begin van het programma, gaat zitten, terwijl het sorteren zelf zeer snel is. Wat betreft de assembler subroutine, in regel 1 t/m 6 deze heeft boven het verhogen van de snelheid ook nog als voordeel dat nu de gehele string 'meetelt' bij het sorteren. Dit laatste in tegenstelling tot Bubble-sort, waarbij alleen de eerste zoveel karakters meetellen.

Sorteren. "SRT9 A.M." ****

Sorteerprogramma, SRT7 van A.Marchal bewerkt en toegelicht door Borghaerts. Het programma laden met *LOAD, het gaat dan naar ≠8200. Bij de 1e keer ?18=≠82 END RUN dit om de vector goed te zetten voor het DIM statement. Het bestand komt bv. op ≠2900 (ingetypt of geladen van band). In het laag geheugen, ≠2900 tot ≠4000, gaan ruim 100 volledige namen en adressen met telefoonnr. Als tussenopslag wordt de videoruimte gebruikt.

Voorbeeld van gebruik SRT9 voor achtereenvolgens SORTEREN en daarna VERTIKAAL getabellerd uitprinten van 'n adresbestand.Per lijnnr. tot 32 karakters, met komma's verdeeld in max. 7 (sub)strings. Voor HORIZONTAAL uitprinten de linefeed's (P.') verwijderen. De variabele C staat voor (sub)string-nr. Invoer van P.$14 achter substring-nr. geeft Vet-Drukken van de daaropvolgende substring. P.$15 heft dit weer op.

```
10 AB,JANSEN,MEIDOORNWEG 4,1010XY,PIEPERVEEN           }b.v. op
20 CD vd,BERG,JAVASTRAAT 1,2000PQ,BOBBELDORP           } ≠2900
30 EF uit den,BOOGAARD,PARKLAAN12,5555VW,GASSEGTERNIJVEEN }
?18≠82                (sprong naar sorteerprogram)
RUN
SORTEREN OP LIJNNR.? ,TYPE 0
OP ALFABET 1E STRING?TYPE1
OP ALFABET 2E SRING? TYPE 2 ENZ
U KIEST...?2          (achternaam=2)
TE PRINTEN SUBSTRINGS NRS?0    (=alle strings of type 12345)
MET LIJNNR. ?J/N?N
    CD vd BERG                 )afstand staat op lijnnr.792
    JAVASTRAAT 1               )afstand staat op lijnnr.832
    2000PQ BOBBELDORP          )afstand staat op lijnnr.834
    EF uit den BOOGAARD        )afstand staat op lijnnr.852
    PARKLAAN12                    Kantlijn afstand staat op lijnnr.793
    55555VW GASSELTERNIJVEEN              en in subroutine f
    AB JANSEN
```

Handig kan zijn:
regel 34 P.$3 printer uit
regel 791 P.$2;P."DRUK TOETS";
LINK≠FFE3 printer weer aan

Eprom-programmeren

WERKEN MET DE ZERO PROGRAMMER:

De Acorn Atom heeft een zgn. Vrije Socket IC24 waarin naar believen 4K ROM's of EPROM's gestoken kunnen worden. De ROM's komen van de fabriek, de EPROM's worden zelf geprogrammeerd, bijv. de 2732 (via tussenvoet) of de 2532. Deze laatste kan zonder meer in de socket IC24, het adres is ≠A000-≠AFFF.
Het snelste gaat u te werk wanneer u bij bestellen van de programmer bij ZERO opgeeft: 1)Welke EPROM u gaat gebruiken en 2)Dat de te leveren software dient voor de Acorn Atom. U ontvangt dan een 'startklare' versie van de ZERO-software voorzien van de goede routine-adressen van de Atom. Deze software zit dan in de door u gekozen EPROM, die u na eerste gebruik kunt wissen en hergebruiken.
De eerste taak: De in EPROM geleverde software moet naar Atom-RAM en vandaar naar cassette. Steek ZERO-EPROM in socket IC24 ('n 2732 met tussenvoet). Dan typt u in: LINK≠A7B9, dan returntoets en wacht af. De software schrijft dan over naar RAM ≠8200 en hoger. Wanneer dit (in 20 sec) klaar is, schrijft u RAM naar cassette met: *SAVE"ZERO"8200 8A00. De cassette is voor 2e en verder gebruik. De ZERO-EPROM kan eruit, gewist en er kan iets anders in.
De tweede taak: Uw in EPROM te zetten program moet eerst naar uw RAM ≠2900 en dan naar de programmer. Te doen: Haal computer uit stopcontact. Steek uw te programmeren EPROM in de ZERO-PROGRAMMER (in goede richting) altijd zonder tussenvoet. Steek flat-kabel-connector van de ZERO-PROGRAMMER in IC24 voet, ook altijd zonder tussenvoet en goede richting. PROGRAMMER op 5 Volt! Van cassette haalt u de ZERO-software terug naar ≠8200 met: *LOAD"ZERO"8200. (Was weg door uitzetten van computer.) Uw eigen programma, dat u in EPROM wilt zetten typt u in (vanaf ≠2900 dus) of vanaf cassette met: LOAD"EIGEN", gaat dan automatisch naar ≠2900. Van dit EIGEN program zoekt u de TOP met: PRINT&TOP en krijgt XXXX d.i. hoogste adres eigen program, plus 1 byte.
Nu volgt het programmeren van EIGEN EPROM. In RAM zit vanaf ≠8200 ZERO-soft en vanaf ≠2900 uw program. Start de ZERO-soft met: LINK≠8200. Het ZERO-soft begint nu een conversatie. Op de eerste pagina van die conversatie controleert u of ZERO het goede EPROM-typenr. geeft. ZO NIET: dan op ZERO-PROGRAMMER het schakelmoduul goedzetten. Zie tek. handleiding. Controleer ook of als byte-lengte (ZERO-soft) nu EF wordt gegeven. ZO NIET: dan zit er een fout in de ZERO-soft (niet waarschijnlijk) of is er iets fout gegaan bij het overschrijven van de ZERO-soft naar uw ≠8200. Zie 'eerste taak'. U kunt taak 1 overdoen of 'anders' doen n.l. met een verderop nog te geven hulp-programma'tje.
Nemen we aan dat de controles goed zijn, dan vult u de conversatie in met: BASE ADRESS . . . A000 en bij CHOOSE NOW . . . 4 FIRST PROM BYTE . . . 0 en LAST PROM BYTE . . . FFF bij BYTE COUNT . . . gewoon return. Doet hij zelf.
Blijkt uw EPROM inderdaad leeg, dan kiest u CHOOSE NOW . . . 2 en vult in:(1) FIRST MEM BYTE . . . 2900
(2) LAST MEM BYTE . . . XXXX-1
(3) BYTE COUNT . . . (returntoets)
(4) FIRST PROM BYTE . . . 0 (niet A0!!)
(5) LAST PROM BYTE . . . (returntoets)

Volg verder de conversatie. Het programmeren duurt 100sec. Wilt u (later) uw EPROM 'bijvullen' dan wordt (4) natuurlijk het eerste vrije byte-adres.

Hulp-programma'tje voor overschrijven en controleren (resp. regel 10 en 20). N.B. Het Hulp-program is buiten de haakjes algemeen geldig. Het 'loopt' uiteraard met RUN. Het foute BYTE kunt u peeken en pooken. Lukt dit niet, dan is de RAM stuk.

```
 5 INPUT "STARTADRES Z" A      (of A=≠A000)  )
 6 INPUT "EINDADRES Z" E       (of E=≠A7FF)  ) ZEROsoft is 2K lang
 7 INPUT "STARTADRES PROGR" D  (of D=≠8200)  )
10 E=E-A; FOR I=0 TO E; D?I=A?I
20 IF D?I<>A?I GOSUBa
30 NEXT; END
40aP."FOUT IN BYTE"I
50 RETURN
```

AANPASSING van de ZERO EPROM programmer, zodanig dat er 2764 EPROM's mee geprogrammeerd kunnen worden.

Benodigheden: 1 28 pins socket
1 schakelaar dubbel om
1 schakelaar enkel om
1 zener 4V7 400mW
1 zener 5V6 400mW
1 spannings regelaar 78L15

Indien uw programmer reeds met een 78L15 is uitgerust, dan zijn de laatste 3 componenten niet nodig.

U vondt naast het IC NE555 eerst 4 weerstanden en dan 3 diodes. De eerste diode vervangt u door een zener van 4V7 (streepje richting programmeer-socket) en de tweede diode vervangt u door een zener van 5V6 (richting idem). Vervolgens vervangt u het component direct naast de schakelaar (78L24) door een 78L15. Het soldeer-eilandje aan de onderkant van de print, onder de eerste zener van 4V7 kan nu doorverbonden worden voor 2732A en 2764 EPROM's, of open gelaten worden voor 2716, 2732 en 2532 EPROM's.

Verwijder nu het jumper IC en vervang dit door onderstaande schakeling, incl. nieuwe 28 pin programmeeervoet. De A12 schakelaar is om de te programmeren helft in te stellen. Schakelaar S2 moet in de stand "read" staan. Programmeren: bij de melding "switch Vpp to 25 volt", eerst S2 in de stand "prog", daarna de altijd gebruikte schakelaar in de stand "25 volt" zetten en dan pas de spatiebalk indrukken. Bij terug schakelen: eerst 25 volt afschakelen, daarna S2 omzetten en pas daarna spatie geven. Het is normaal dat de programmer in de 2732 mode staat.

The Acorn Atom has a vacant expansion socket inside for a utility ROM which will accommodate up to a 4K by eight-bit device, writes John Flower of Cowplain, Hampshire. The EPROM which can be fitted in the 2532 which is made by Texas Instruments and a number of Japanese suppliers. This device is a 32K chip organised as 4,000 words of eight bits. The chip is readily available for about £6.

You can program your favourite game or a useful machine-code routine on to this device so that it can become a permanent part of your computer's operating system. You could even write your own toolkit program. The 2532 EPROM is remarkably easy to program with the Atom since most of the necessary circuitry for an EPROM programmer is already inside the Atom's VIA chip. If you decide to have a go at building this design then you will have to buy and fit the optional 6522 versatile interface adaptor chip to your machine.

For a programmer interface you only need to provide an address latch externally to address each location in turn while the data is presented and the programming operation takes place. The circuitry consists of 2 LS374 eight-bit latches to hold the address word plus a zero-insertion-force socket to carry the EPROM without risk of bending the pins.

Pin 21 of the 2532 EPROM is connected to +5 volts when reading data. Data is read by pulsing pin 20 low while looking at the data pins. The device works in reverse if pin 21, which is normally held to 5 volts, is taken to 25 volts. In this case data present at the data pins is programmed into the eight locations whose address is present at the address pins. This happens if pin 20 is pulsed low for exactly 50ms.

The program performs the necessary operations to copy, program and verify EPROMs. Programs to be copied from an EPROM or to be programmed into one are stored in the graphics memory from location #8400 onwards. The program is menu driven and prompts to see what operations you wish to perform.

The VIA chip writes the relevant address on to ports A and B and clocks the address latch to store the 12-bit address. Then the eight-bit data is either presented to or read from port B.

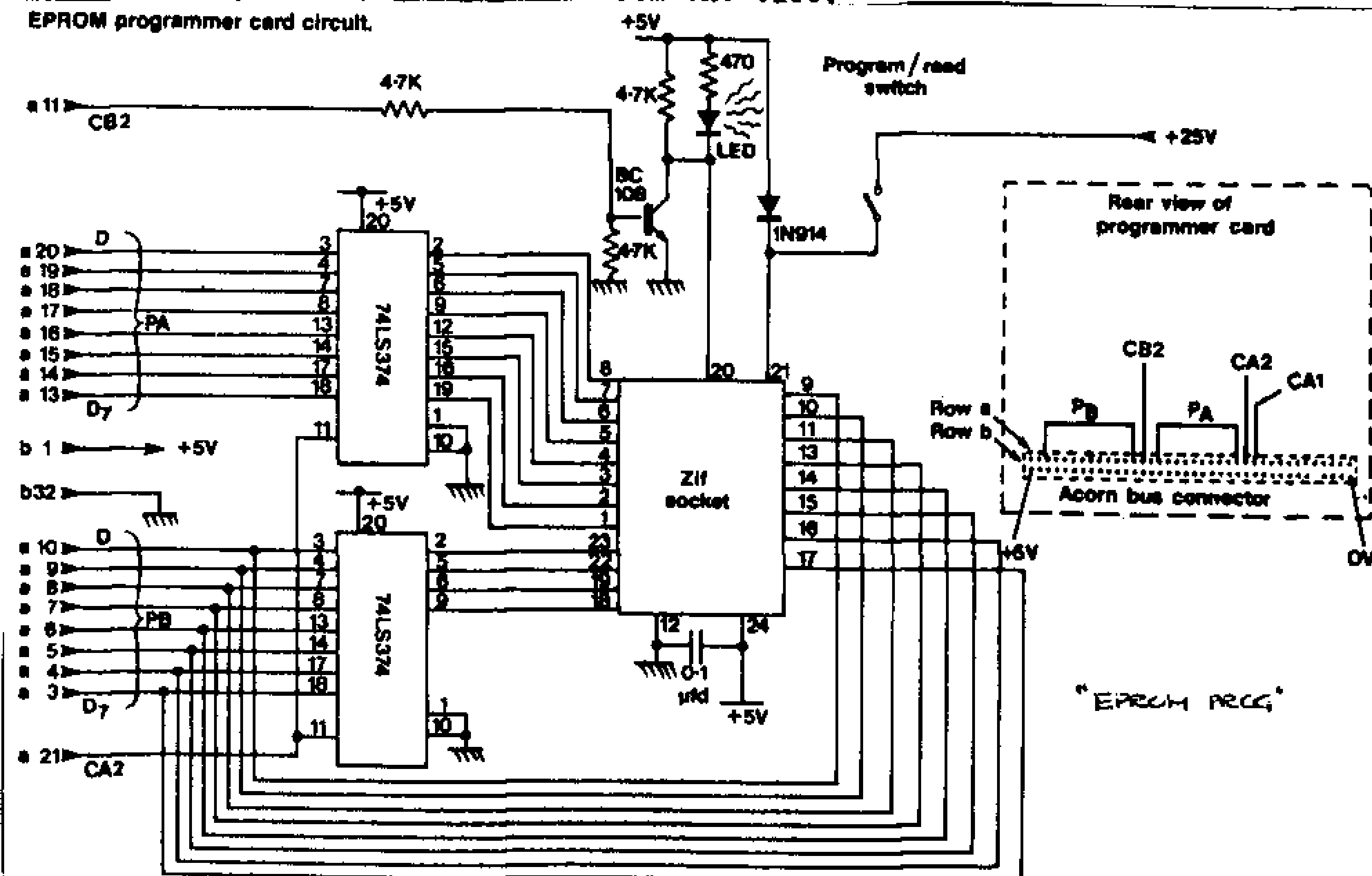
Construction of the circuit should present little difficulty. Only two integrated circuits are involved, plus four resistors, two diodes, an npn transistor and one capacitor. You will need to fit an Acorn bus connector to your computer and buy the appropriate 64-way Eurocard DIN connector mating socket.

The circuit diagram shows the bus connections that are used, viewed from the rear of the programmer card. The 64-way right-angle plug is fitted to the Atom, and the socket is fitted to a piece of Veroboard upon which the programmer is to be constructed.

The light-emitting diode serves to show that Read or Write operations are taking place. The other diode - 1N 914, or similar - ensures that +5V is connected to pin 21 of the EPROM when normal reading of the EPROM occurs. When +25V is applied to the device for programming purposes, the diode becomes reverse-biased so that the 5V power supply is not affected. It is very important to connect this diode the right way round to avoid serious damage to the Atom's +5V supply. The cathode is usually marked by a thick painted ring on the diode body and should be connected to pin 21 and the programming switch. After constructing the circuit you do not need the programmer card to prove that the program is working: with the card unplugged the program will copy #FF into each screen RAM location. You can check this by first

clearing the mode 4 screen, then adding line 95   ?≠B000= ≠F0
Running the EPROM copy routine will let you see the memory being filled, which takes about 4 min. The verify routine reads the EPROM and compares it with the contents of the screen RAM. Any errors will be listed and a simple check-sum is computed by adding the decimal contents of each location together. See if the answer for the all-≠FF case comes to 4,096x255.
Having satisfied yourself that the program works plug in the programmer card and set the Read/Program switch to Read, connection to +25V open-circuit. The diode has been connected correctly if you can measure 5V on pin 21 of the ZIF socket.
The LED should flash during the Copy, Verify and Program modes. If it does not, it may be connected around the wrong way, so try reversing it.
Next insert line 95 to watch the memory being loaded with the copy routine. If you short-circuit pin 20 to each data pin in turn on the ZIF-socket, you can see from the bit pattern whether each bit is being read.
Do not try to fit an EPROM or connect 25V until you are satisfied that everything is in order.
Try reading an EPROM or ROM. You could unplug and read the floatingpoint ROM if you have a fully expanded Atoms. When you are ready to program a 2532 EPROM you can connect a 25V supply to the programmer card: four 6V transistor radio batteries connected in series are a simple way of obtaining it.
If you follow the program instructions to the letter, there should be no problem in programming your own EPROMs, and when you can program EPROMs, you will soon want to erase them. Unfortunately there is no simple method of doing so: intense ultraviolet radiation at a wavelength of 275.8nm, is required.
If you make your own eraser it is essential to mount the tube in a light-thight box to protect the eyes. EPROMs to be erased should be exposed for about 20 min. about 3 cm. from the tube.

EPROM programmer card circuit.



Asteroïden, vliegtuigen en maanlanders. "PLOTTER" ****

Het in beeld brengen van deze en veel andere figuren is steeds opnieuw een probleem. We zouden de Atom Basic commando's PLOT, DRAW en MOVE kunnen gebruiken. Dit kost niet alleen veel schrijfwerk en geheugenruimte, bovendien is de plotsnelheid gering. Vloeiende bewegingen zijn op deze manier onmogelijk. Alleen assembler routines komen in aanmerking. Omdat niet iedereen gewend is om routines te schrijven, heb ik (M.Janssen) besloten een algemeen bruikbare plotroutine te schrijven.
Het enige wat men nu nog hoeft te doen, is het definieren van het gewenste figuur en de subroutine aanroepen. Uiteraard heeft de routine nog een aantal gegevens nodig, zoals de plaats, 'kleur' van de figuur etc. Dezelfde routine kan binnen het hoofdprogramma gebruikt worden om verschillende figuren te plotten, en hoeft dus niet voor iedere figuur opnieuw te worden geassembleerd.
Voor het definieren van de figuur gaat men als volgt te werk. Teken de figuur op ruitjes of millimeterpapier, zoals in fig.1, voor een hartje. Ernaast is aangegeven hoe de figuur wordt afgebeeld op een matrix van 'enen' en 'nullen'. In dit voorbeeld bestaat de fig. uit 7 rijen van alk 9 beeldelementen ofwel 'pixels'. Alleen de pixels waarvoor het matrix-element '1' is, worden door de plotroutine getekend. De rest blijft op het scherm onveranderd.
Zet nu de rijen enen en nullen achter elkaar te beginnen met de bovenste rij en verdeel de rij in groepjes van 8 bit. In ons voorbeeld wordt dit:
01100011 01001010 01100010 00101000 00100010 00100000 10100000
0010000X (het laatste bit 'X' hoort niet meer tot de rij)
Vertaal ieder groepje in een hexadecimaal getal en sla deze getallen op in opeenvolgende geheugenplaatsen, bv. door gebruik te maken van 'poke' instructies. In ons voorbeeld zou dit als volgt kunnen:

```
DIM M(7)
M?0=≠63; M?1=≠4A; M?2=≠62; M?3=≠28; M?4=≠22; M?5=≠20; M?6=≠A0; M?7=≠20
```

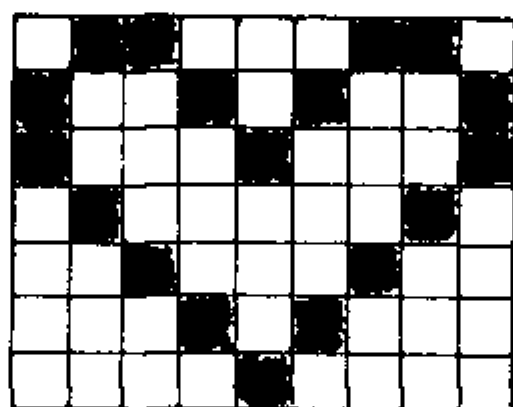
Met woordvectoren i.p.v. bytevectoren kan het nog iets kompakter:

```
DIM M(7)
!M=≠28624A63; M!4=≠20A02022     (denk aan de omgekeerde volgorde)
```

Hiermee is de figuur gedefinieerd en kan hij vervolgens op elke gewenste plaats en in de gewenste 'kleur' getekend worden. Aan de afmetingen van de figuur zijn geen grenzen gesteld.
Om de plaats van de figuur op het scherm aan te geven, gaan we uit van een rechthoekig coordinatenstelsel, waarvan de oorsprong (in tegenstelling tot Atom Basic!) gekozen wordt in de linkerbovenhoek. Dit laatste is om programmeertechnische redenen zo gekozen. De plaats van de figuur wordt nu bepaald door de coordinaten van de linkerbovenhoek van de fig. (zie fig.2)
Voordat de plotsubroutine wordt aangeroepen moeten de volgende geheugenlocaties gezet zijn:

- ≠B000 moet één van de waarden ≠50, ≠90 of ≠D0 bevatten, om aan te geven dat er geplot moet worden in de graphic mode 2a, 3a of 4a respectievelijk. Zie 'Atomic Theory & Practice' 11-6-2.) Het oplossend vermogen is in de X-richting 128 en in de Y-richting resp. 64, 96 en 192.
- ≠80,81 moeten het adres van de figuurgegevens bevatten (≠80=minst significante byte, ≠81=meest significante byte).
- ≠82 en ≠83 dienen de breedte resp. de hoogte van de figuur te bevatten. Voor de fig. in fig.1 is dit 9 resp. 7.
- ≠84 en ≠85 worden gebruikt om de X resp. Y coordinaat van de figuur aan te geven.
- ≠86 dient één van de volgende besturingscodes te bevatten:
    - ≠00 = inverteer figuur ≠80 = plot fig. in wit
    - ≠40 = plot fig. in zwart ≠C0 = plot fig. in grijs

(in grijs bv. om fig. uit te wissen)
De inhoud van bovengenoemde locaties wordt niet gewijzigd door de plot-routine, hetgeen erg handig is bij het herhaald tekenen en uitwissen van dezelfde figuur (bv. voor bewegende figuren).
Tenslotte wordt nog vermeld dat de geheugenlocaties #87 t/m #91 worden gebruikt als tijdelijke opslagplaats van gegevens en dat de routine vanuit Basic wordt aangeroepen d.m.v. LINK SSO.



```
011000110
100101001
100010001
010000010
001000100
000101000
000010000
```

Fig.1

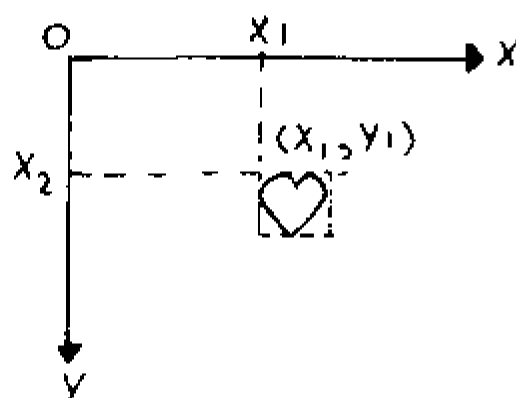


Fig.2

Graf-rite. "GRAF-RITE" ****

Dit is een utility-programma, dat het mogelijk maakt, om met een uitgebreide Atom tekst op het scherm zichtbaar te maken in graphics mode 4. Dit kan zowel bij input als door het key-board.
Met enige aanpassing is het mogelijk dit programma ook te gebruiken in de andere graphics modes. Hierbij moet u wel rekening houden met het feit dat een gedeelte van de karakterset verloren gaat, door een vermindering van het beschikbare geheugen. In deze vorm is het formaat 21 regels van 32 karakters.
Alle karakters, welke via het key-board mogelijk zijn, kan men zichtbaar maken, inclusief kleine letters. Er zijn ook enkele karakters via het key-board ter beschikking, welke normaal niet via het key-board mogelijk zijn: shift [= ←
shift] = →
shift ^ = ↓
shift @ = —
shift \ = +

Aanvullingen:

```
500 F.$12;DIMW64          Clear screen and DIM-string
505 IN."ENTER STRING"$W   Enter string for generation
510 CLEAR4
515 F.M=0 TO LENW-1       Set up loop
520 GOS.(W?M);GOS.a       Set op word vector of Mth char. in string
525 L=K+1;K=L;N.          Reset variables              and print it
```

Listing 1:Generating a string of char. on the screen under Graf-Rite
Before running change line 5 to 5 G.500

```
400 C=0;D=0;F=0;G=1
405 IF G=1 F=C;G.415
410 F=D
415 STR F,B
420 IF F<10 GOS.CH"0";GOS.a;
    L=K+1;K=L
425 GOS.(B?0);GOS.a;L=K+1;K=L
430 IF F>9 GOS.(B?1);GOS.a;
    L=K+1;K=L
435 IF F>99 GOS.(B?2);GOS.a;L=K+1;
    K=L
440 IF G=1 C=C+1;L=92;K=L;G=2;G.410
445 D=D+1;L=64;K=L;G.405
```

Listing 2:A score indicating routine that can be added to games etc.
Before running change line 5 to
5 CLEAR 4;G.400

Universele inverteer-routine. ********

Deze universele routine inverteert het aktuele beeldscherm (graphics of tekstmode). Er wordt een soort binair boompje gebruikt om het eindadres vast te stellen. De vertaalde code is 70 bytes lang, gebruikt alleen de geheugens ≠90 en ≠91 (te vervangen door elk ander opvolgend 'ZP-paar') en wordt aangeroepen met LINK LL0.
(Het statement INVERT, dat in de handel is, doet hetzelfde.)

```
:LL1 \TEKST
 LDA≠8000,Y
 EOR@≠80
 STA≠8000,Y
 LDA≠8100,Y
 EOR@≠80
 STA≠8100,Y
 INY;BNE LL1;RTS
:LL0 LDY@0;STY≠90
 LDA≠B000;BEQ LL1
 ROLA;BCC LL2
 LDX@≠8B
 ROLA;BCC LL3
 LDX@≠97;BNE LL3
:LL2
 LDX@≠83
 ROLA;BCC LL3
 LDX@≠87
 ROLA;BCC LL3
 LDX@≠85
:LL3 STX≠91 \GRAPHICS
:LL4
 LDA(≠90),Y
 EOR@≠FF
 STA(≠90),Y
 INY;BNE LL4
 DEX;CPX@≠80
 BPL LL3;RTS
```

Graphics-inverteer-programma 1.
Het graphics geheugen (de volle 6K) wordt door deze routine X keer geïnverteerd. Als X=0 gebeurt dat 256 keer en duurt 26 sec. De vertaalde code is 31 bytes.

Graphics-inverteer-programma 2.
Ook hier owrdt de volle 6K graphics geinverteerd. Voor X=0 duurt het nu 18 sec. De vertaalde code is 204 bytes.

Scherm-inverteer-programma.
Dit programma inverteert het normale scherm (mode 0). Ook hier gebeurt dat X keer.

```
10DIM LL1,P-1
20[:LL0 LDY@0
30 LDA@≠80;STA≠91
40 LDA@≠00;STA≠90
50:LL1
60 LDA(≠90),Y
70 EOR@≠FF
80 STA(≠90),Y
90 INY;BNE LL1
100 INC≠91;LDA≠91
110 CMP@≠98;BNE LL1
120 DEX;BNE LL0;RTS;]
130 END
```

G-i-p 1.

```
10DIM LL2
20F.I=1 TO 2;DIMP-1
30[:LL0 LDY@0;:LL1;]
40F.G=≠8000 TO ≠97FF
  S.≠100
50[LDA G,Y;EOR@≠FF;STAG,Y;]
60NEXT G
70[INY;BNE LL2:DEX;BNE LL2;
  RTS;:LL2 JMP LL1;]
80NEXT I
90END
```

G-i-p 2.

```
10DIM LL1,P-1
20[:LL0 LDY@0
30:LL1
40 LDA≠8000,Y;
   EOR@≠80
50 STA≠8000,Y
60 LDA≠8100,Y;
   EOR@≠80
70 STA≠8100,Y
80 INY;BNE LL1
90 DEX;BNE LL0;
   RTS;]
100END
```

S-i-p.

Bram Poot.

Noise vrij tekenen. *******

Door A.Reijnen werd een programma'tje uit Byte ingezonden. Toelichting:
Na CLEAR X in het programma intoetsen:

```
?≠80=?≠3FE;?≠81=?≠3FF
?≠3FE=TOP&≠FF;?≠3FF=TOP/256
A=TOP
!A=≠22C80A9
A!4=≠6CFBD0B0
A!8=≠80
```

Het tekenen van punten en trekken van lijnen gebeurt nu zonder interferentie op het scherm, maar gaat nu trager. Vertraging per punt 1/60 sec.

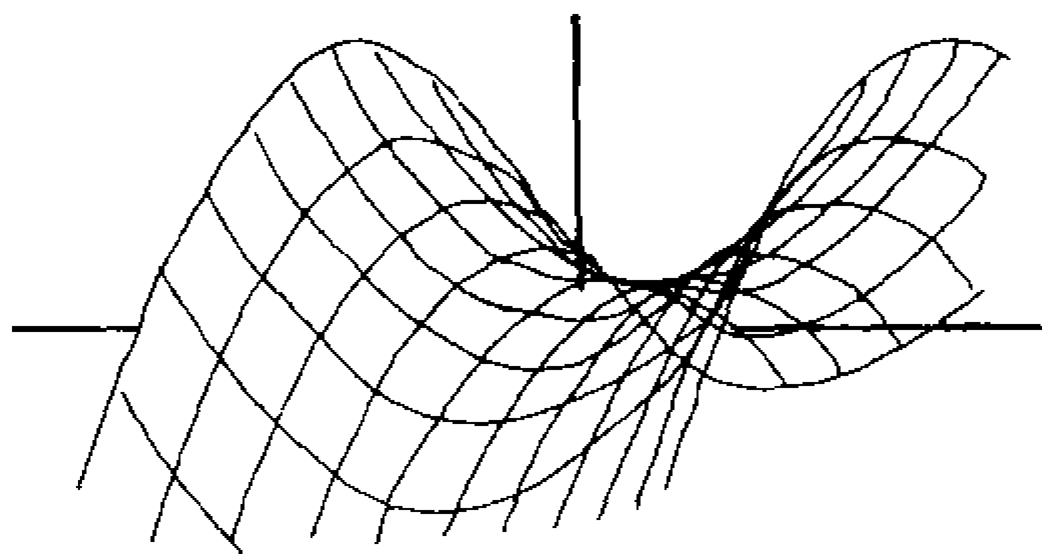
3-D plot. "3-D PLOT" *****

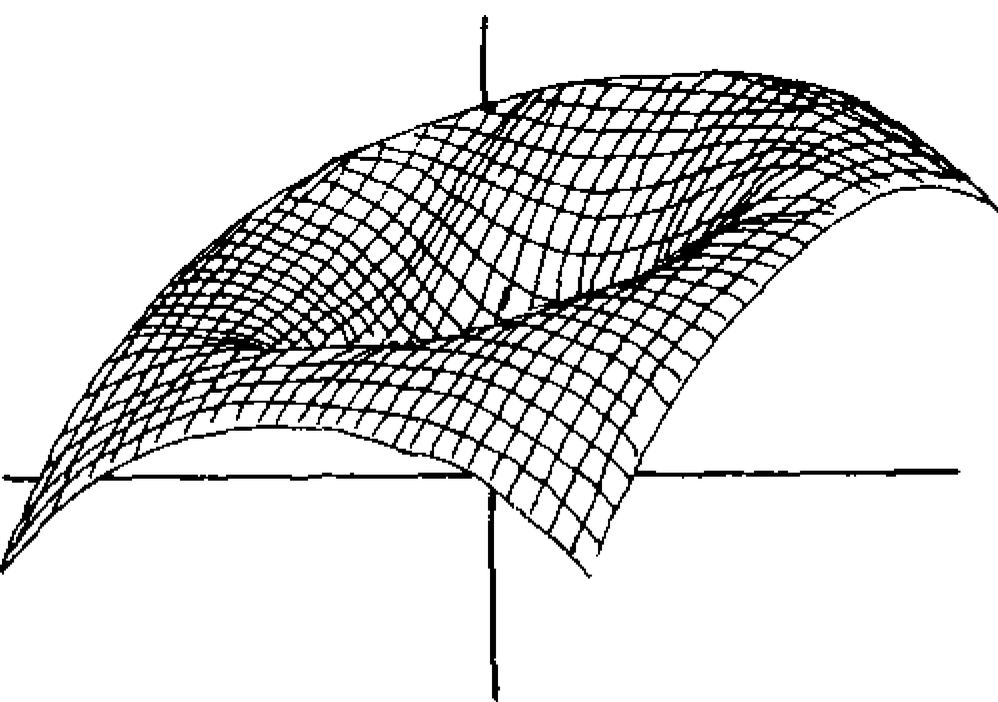
Plotje ingezonden door R.Heuvel. (Uit Personal Computing.)

3-Dimensioneel tekenen. "3D PLOT/PRINT" *****

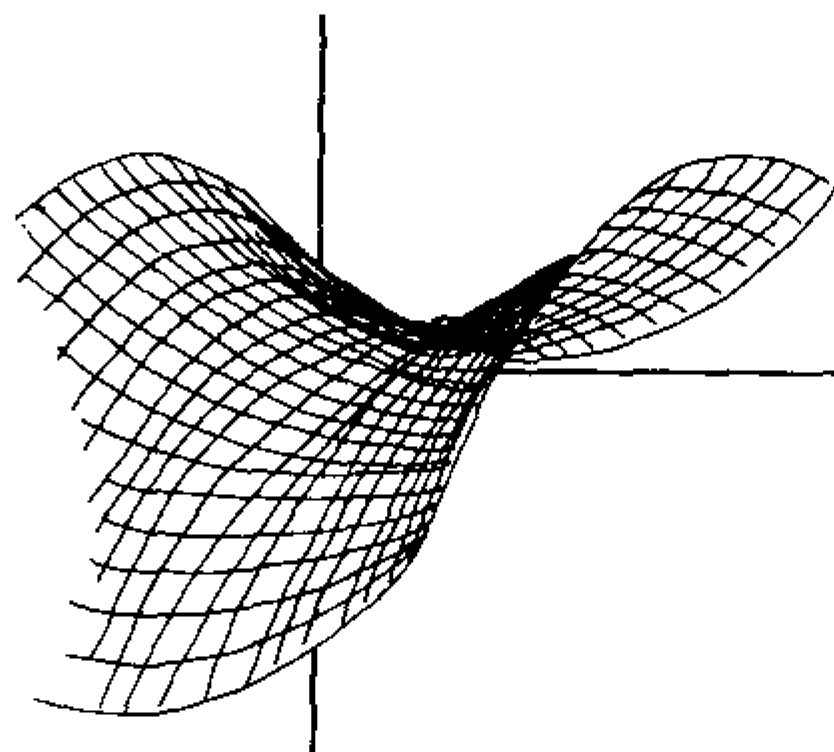
Het 3-Dimensioneel tekenen van wiskundige vergelijkingen en de projectie hiervan op het scherm incl. het uitwissen van de zgn. 'Hidden Lines', volledig met een (j/n) printroutine. Door Jos Horsmeier.

```
 10 VOOR DIT VOORBEELD INTYPEN:
 20 X-MIN: -1        Y-MIN: -1
 30 X-MAX:  1        Y-MAX:  1
 40 F-MIN: -1        F-MAX:  1
 50 X-TTL: 10        Y-TTL: 10
 60 INCLT.ANGLE (DEG): 45
 70 F(%X,%Y)= %Y*%Y-%X*%X
 80 VISABLE AXES(Y/N): Y
 90 PRINTOUT  (Y/N):   Y
110 WACHT DAN TOT TEKENING OP 'T
115 SCHERM KLAAR IS (PIEPJE) EN
120 DRUK DAN EEN VAN DE TOETSEN
```





F(%X,%Y)=SIN(SQR(%X*%X+%Y*%Y))



F(%X,%Y)=%Y*%Y-%X*%X

Het programma is geschreven naar de 8255-C3.

Spelletjes en muziek

Tarantella Napoletana. "MUSIC" *******
Frans van Hoesel laat uw Acorn 'Tarantella Napoletana' spelen. U heeft er wel de Toolkit voor nodig.

Muziek uit het noorden. "MUZIEKDOOS" "WILHELMUS" ********
Twee muziek programma's van Wim Schreuder, voor iedereen met een Toolkit. Het zijn 'Muziekdoos' en 'Wilhelmus van Nassouwe'.

Toonhoogte:

		oktaaf1	oktaaf2	oktaaf3	oktaaf4
DO	C		205	102	50
	C		194	96	47
RE	D		183	91	45
	D		173	86	42
MI	E		163	81	39
FA	F		154	76	37
	F		145	72	35
SO	G		137	68	33
	G		129	64	31
LA	A	246	121	60	29
	Bb	231	114	57	
TI	B	217	108	54	26
DO	C				24

Toonlengte

sec.	getal
5	250
2	100
1	50
½	25
1/4	13
1/8	6
1/16	3

Spatie: van bv. 2 sec:
BEEP X,Y waarbij:
Toonhoogte =1
Toonlengte =100

Bingo. "BINGO" ******
Programma van Wim van Ham.

Lotto-hulpje. "LOTTO" ********
Dit programma verzint getallen, die je kan invullen op je lotto-formulier. Auteur: G.Akkermans.

Geintje. "GEINTJE" *********
Van Gert de Jager is dit programme'tje. Kunt u ontdekken waarom het programma'tje doet, wat het doet?

Grapje. *****
Grapje van Jos Horsmeier. Wilt u iets laten knipperen? Typ dan in:

```
10 ?≠E1=0
20 P."KNIPPEREN"
30 P.$13;LINK≠FE22;GOTO 20
```

Het knippert wet vlug, maar daar kunt u met wat 'WAIT's wel wat aan doen.

Joedel. "JOEDEL" *********
Snelle joedel. Een aanvulling op regel 85 kan zijn DEC≠80.

Nimspel. "NIMSPEL" ********
Nimspel is een denkspelletje dat je tegen de computer speelt. Je hebt er de TOOLBOX voor nodig en Gerard Akkermans heeft het gemaakt.

Afronden. "AFRONDEN" ******

F. van Hoesel stuurde een programmaatje in om goed af te ronden. Een probleem bij veel mensen. Toelichting: het aantal spaties in het PRINT statement van regel 20 is gelijk aan @-2.

Afronden 2. ******

Ook J.Jobse stuurde een versie in van zijn afrond-programma.
Toelichting op FIX routines:
Naamgeving: De naam 'FIX' is afkomstig uit de programmeertaal ALGOL, waarin dit statement een soortelijke functie heeft.
Toepassing: FIX2 is ontwikkeld voor programma's waarin geld een rol speelt. FIX F is universeel toepasbaar en geeft een nette en overzichtelijke uitvoer van FP-getallen.
Gebruik: Het printen van %A op 4 decimalen nauwkeurig gaat als volgt:
F=4; %X=%A; GOSUB f
Opmerkingen:- de waarde van @ heeft een iets afwijkende uitwerking. @ bepaalt waar de eenheden geprint worden.
- met komma wordt steeds de 'decimal point' bedoeld. Bij geldbedragen kan deze in FIX2 e.t. vervangen worden door een echte komma
- 5 en hoger wordt naar boven afgerond.

Voorbeeld: FIX F

```
10 F=3;@=6;P.'                  RUN
20 %X=0;GOS.f;P.'                 0.000
30 %X=-1/3;GOS.f;P.'             -0.333
40 %X=10000/6;GOS.f;P.'        1666.667
50 %X=9.9994;GOS.f;P.'            9.999
60 %X=999.9995;GOS.f;P.'       1000.000
70 %X=-1/1000;GOS.f;P.'          -0.001
80 %X=-0.0004;GOS.f;P.'          -0.000
90 %X=PI;GOS.f;P.'                3.142
100 END
```

FIX2: Begrensing op 2 plaatsen achter de 'KOMMA'

```
1000fREM FIX 2
1003 %@=@;Z=%(100*ABS%X+0.51)-ABS%X*100;IFZ>99P.%X+SGN%X".00";R.
1005 FIF%X<OF IF%X>-100 P.$9;U.C.>=@-2;P."-";@=0
1006 P.%X".";@=0;IFZ<10;P.0
1007 P.Z;@=%@;R.

1000fREM FIX 2N
1001 FIF ABS%X<0.005;FIF%X<>0;P.$7,0".00";R.
1002 FIF ABS%X>2↑31/100;P.$7;@=@+2;FP.2↑31/100,"0";@=@-2;R.

1000fREM FIX 2F
1001 FIF ABS%X<0.005;FIF%X<>0;FP.%X;R.
1002 FIF ABS%X>2↑31/100;FP.%X;R.
```

Gebruikt: Label f, invoer-variabele %X en @ (na aanroep ongewijzigd),
hulp-variabelen %@ en Z (na aanroep ongedefinieerd)
Beperkingen: ABS(%X) max. = 21474636 ABS(%X) min. = 0.005

FIX F: Begrenzing op F plaatsen achter de 'KOMMA'

```
1000fREM FIX F
1001 %@=100*F+@;F=%(10↑F+1)
1004 Z=%(F*ABS%X+0.51)-ABS%X*F;IFZ=F;P.%X+SGN%X".";Z=0;G.1007
1005 FIF%X<0;FIF%X>-1;DO P.$9;U.C.>=@-2;P."-";@=0
1006 P.%X"."
```

```
1007 @=0;DO IF Z<F/10;P.0
1008 F=F/10;U.F=1;IFZ>0;P.Z
1009 F=%@/100;@=%@%100;R.

1000fREM FIX FM
1002 FIF ABS%X<0.5/F;FIF%X<>0;P.$7,0".";Z=0;G.1007
1003 FIF ABS%X>2↑31/F;P.$7;@=@+%@/100;FP.2↑31/F;P."0";G.1009

1000fREM FIX FF
1002 FIF ABS%X<0.5/F;FIF%X<>0;FP.%X;G.1009
1003 FIF ABS%X>2↑31/F;FP.%X;G.1009
```

Gebruikt: label f, invoer-variabelen %X, F en @ (na aanroep ongewijzigd)
hulp-variabelen %@ en Z (na aanroep ongedefinieerd)

Beperkingen: F max. = 8 met ABS(%X) max. = 21.4748364
F min. = 1 met ABS(%X) max. = 214748364

Versies: . geen max./min. beveiliging (snel en kort)
M begrenzing op max./min. waarde (met waarschuwing)
F automatische FP-notatie bij overschrijding max./min.
Bij gebruik van een beveiligde versie de regels toevoegen aan de 'kale' versie.

Talstelsels. "TALSTEL" ********
Programma van Kuckartz, schrijft een binair, octaal, decimaal of hexadecimaal getal in de andere talstelsels om.

Priemgetallen. "PRIEM GETALLEN" *******
Dit programma van G.Akkermans, berekent alle priemgetallen groter dan de opgegeven waarde.

Rekenen. "HOOFD REKENEN" ******
J.Post stuurde een hoofdreken programma in, het is een programma van de PET-gebruikersclub, door hem omgeschreven voor de Acorn Atom.

Hypotheek berekenen. "HYPOTHEEK" ******
Programma van H.Blijleven, dat u helpt uw hypotheek te berekenen.

Sinusteller. *****

```
 20FDIM%TT(120)
 30FORI=0TO120;%TT(I)=0;NEXT
 40CLEAR2;PRINT$21
 43F.I=#8000 TO#8600 S.4;!I=#FFFFFFFF;N.
 45%C=15
 60FORH=1TO28
 80%A=%C/(H/2)
100FORX=0TO120;T=3*X*H;%T=RAD(T)
120MOVEX,0;PLOT5,X,95;MOVEX,45
140%U=%A*SIN(%T)
160%TT(X)=%TT(X)+%U;Y=%(%TT(X))
180PLOT2,0,Y
200NEXT
220NEXT
240END
```

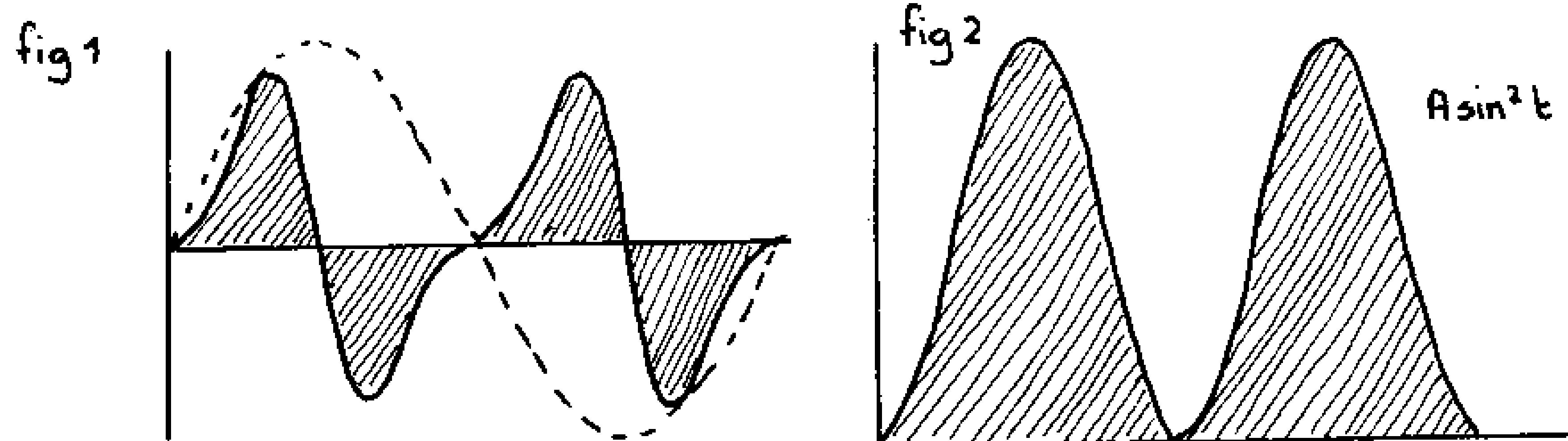
Programma van H.Heuvel, dat bedoeld is als proef om de opbouw van golfvormen uit sinusvormige spanningen te demonstreren. %C is de initiele amplitude van het sig. In regel 60 wordt de orde van de harmonisch bepaald. In regel 80 verder de spanning van de harmonische spanning. De waarden in dit programma zijn voor een ZAAGTANDvorm.

Fourier analyse. "FOURIER" ********
A.Harteveld reageerde op het hierboven staande programma.
Dit programma toont aan dat willekeurige golfvormen opgebouwd kunnen worden gedacht uit sinussen met een frequentie die een geheel meervoud

zijn van de grondfrequentie van de betreffende golfvorm. Deze sinusvormige componenten worden harmonischen genoemd. Wanneer we bv. spreken over de derde harmonische van een golfvorm dan bedoelen we de component die een frequentie heeft van 3x de grondfrequentie. Willen we van een bep. harmonische de grootte en de fase uitrekenen dan dienen we dit te doen door zgn. FOURIER ANALYSE.
De fourier analyse, of in een andere vorm fourier-transformatie genoemd, berekent in principe voor een willekeurige harmonische de grootte (modulus) en de fase. Door herhaald uitvoeren voor verschillende harmonischen ontstaat dan een frequentiespectrum van het periodieke signaal met de gekozen golfvorm. Dit spectrum is een VOLLEDIGE BESCHRIJVING van de originele golfvorm (alleen nu een functie van de frequentie in plaats van een tijd-functie). Door optellen van alle harmonischen in de juiste grootte (modulus) en juiste fase ontstaat weer het originele periodieke signaal.
De fourier-transformatie is een wiskundige truc die eenvoudig te begrijpen is. Een transformatie wordt uitgevoerd door een periode van de grondgolf te vermenigvuldigen met een sinus van de te bereken harmonische frequentie. Vervolgens moeten we dan deze nieuwe functie over de gehele periode van de blokgolf integreren. Meer begrijpelijk gezegt: we gaan het NETTO oppervlak van de ontstane functie bepalen. Het NETTO oppervlak is hierbij dan het oppervlak boven de nul-as minus het oppervlak onder de nul-as. Wanneer we een sinusvormige golfvorm vermenigvuldigen met een hogere harmonische sinus ontstaat bv. een functie zoals weergegeven in fig.1. Het NETTO opp. van dit figuur is nul! In dit geval betekent dat, dat de 2e harmonische van een sinus nul is. Zouden we dit ook doen voor alle hogere harmonischen dan zien we dat ook hier weer nul uitkomt.
Wanneer we echter een sinusvormige golfvorm vermenigvuldigen met een sinus van dezelfde frequentie, dan ontstaat een functie zoals is aangegeven in fig.2. In beide gevallen is nog geen faseverschuiving geintroduceerd, doch dit doet aan het principe niets af. Het NETTO opp. van fig.2 is zoals te zien ongelijk aan nul. De grootte van dit opp. is afhankelijk van de grootte en fase van de betreffende sinuscomponent (we kunnen de sinus waarvan we zijn uitgegaan ook als een component van een gecompliceerdere golfvorm zien). Het zal duidelijk zijn dat m.b.v. deze methode steeds alleen de te berekenen harmonische wordt bepaald, aangezien andere harmonischen geen NETTO bijdrage aan het opp. leveren.
We kunnen dus door steeds met een andere harmonische sinus te vermenigvuldigen en te integreren het frequentie spectrum van een periodiek signaal berekenen. Een harmonische wordt echter niet alleen bepaalt door zijn modulus doch ook door zijn fase. De fase van een harmonische en de daarmee samenhangende invloed hiervan op het "netto-opp.", kunnen we vangen door de bovenstaande methode 2 maal uit te voeren. De 1e maal vermenigvuldigen we met een sinus en integreren en de 2e maal vermenigvuldigen we (de originele golfvorm) met een cosinus om vervolgens weer te integreren. De modulus van de harmonische bepalen we dan door Pythagoras toe te passen op de gevonden componenten. De fase kan bv. dan worden berekend uit: arccos(c/m) waarin c en m resp. de cosinus-component en de modulus voorstellen.

Het programma doet praktisch letterlijk hetgeen hierboven beschreven is. Het vermenigvuldigen met resp. sinussen en cosinussen geschiedt m.b.v. een sinustabel. Dit om te voorkomen dat sinuswaarden die steeds weer als vermenigvuldigingsfactoren worden gebruikt meerdere keren worden berekend, hetgeen veel tijd kost.
In lijnen 510,515,530 en 535 worden o.a. deze sinuswaarden uigelezen.

Het programma bestaat uit 3 delen die ieder aan elkaar gekoppeld zijn via ≠FFE3. Men dient dus voor een verdere programma-executie een willekeurige toets in te drukken. Eerst verschijnen, nadat alle vragen zijn beantwoord, de moduli van de berekende harmonischen. Daarna worden de fasen uitgeprint terwijl vervolgens een grafische voorstelling van het (amplitude)spectrum op het scherm verschijnt. Het opzetten van het source-array kan gebeuren vanaf regel 62 tot regel 98. Gebruik hiervoor %TT (I). De nauwkeurigheid wordt in de eerste plaats door de omvang van het source-array bepaald. Wellicht kan een andere integratie-methode een grotere nauwkeurigheid opleveren (toegepast is de regel van Simpson).



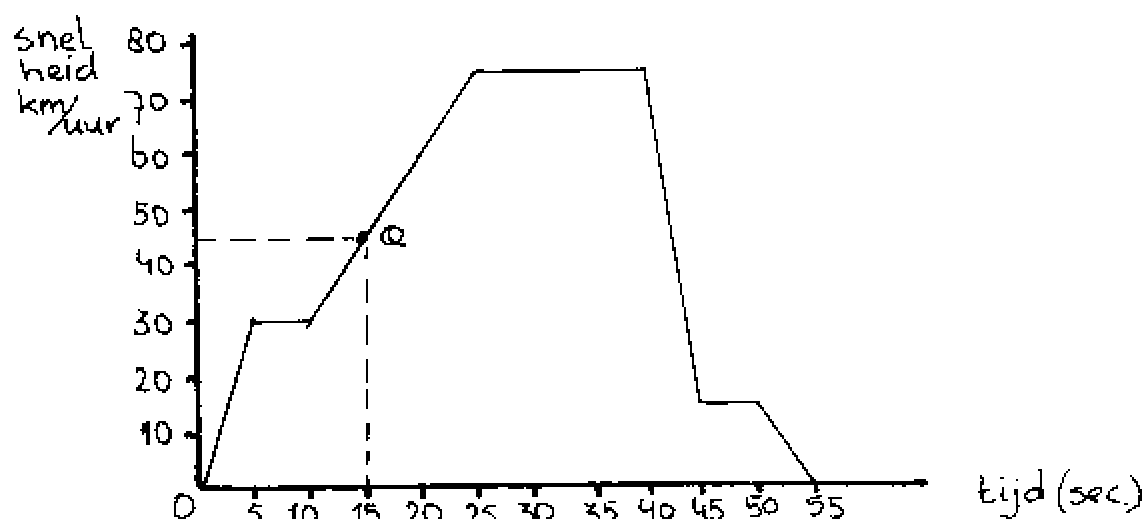
Grafieken. "GRAFIEKEN" *****

Ik zal in dit artikel proberen duidelijk te maken waarom grafieken zo belangrijk zijn in de wiskunde. Vervolgens geef ik een overzicht van gebruikte termen in de wiskunde, om tenslotte te eindigen met een program voor de Acorn, dat grafieken tekent in graphics-mode 4. (FP-ROM nodig.)

Waarom grafieken?

Stel je rijdt in een auto, en om de 5 sec. kijk je op je km-teller wat je snelheid is. Je resultaten van deze waarnemingen kunnen er als volgt uitzien:

tijd/sec	snelh.(km/uur)
0	0
5	30
10	30
15	45
20	60
25	75
30	75
35	75
40	75
45	15
50	15
55	0



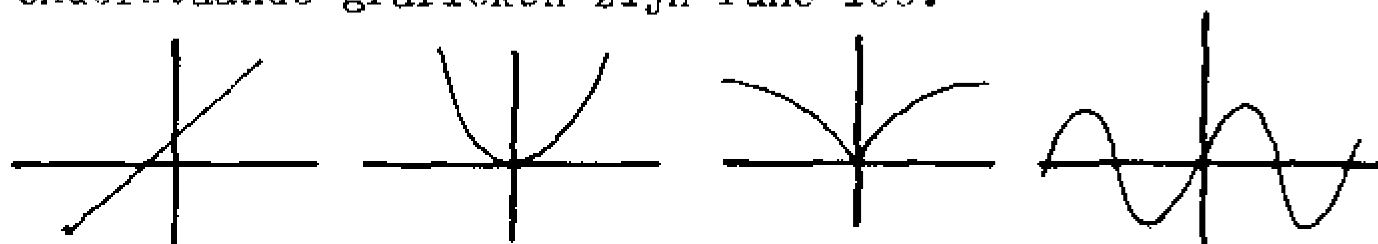
Een tabel is één manier om je gegevens ordelijk op te schrijven. Een andere manier om de gegevens zichtbaar te maken is de grafiek.

Uit de grafiek zijn sneller kenmerken van het rijgedrag te halen dan uit de tabel. Bv. zien we in één oogopslag dat in de periode 5-10 sec. de snelheid konstant bleef evenals in de periode 25-40 sec. Ook zien we in de grafiek heel snel wanneer de auto snelheid verliest, nl. in de periode 40-45 sec. Maar ook meer delicate zaken, zoals de afstand die de auto na bv. 10 sec. heeft afgelegd zijn door een geoefend wiskundige via de grafiek te ontdekken. Ook de remkracht, het acceleratievermogen etc. zijn in de grafiek verborgen. Dit zijn allemaal voordelen die de grafiek boven de tabel heeft. Juist deze punten maken de grafiek erg waardevol.

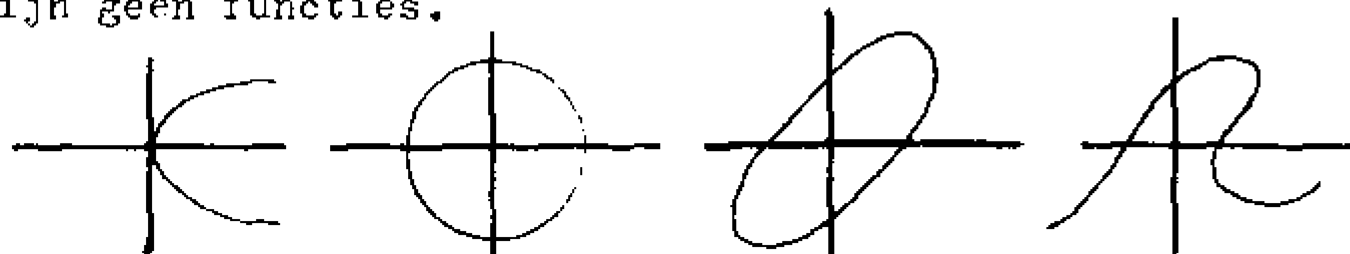
We gaan de zaken wat abstracter maken.

De horizontale as in de grafiek noemen we de X-as, omdat hierlangs de variabele 'X' wordt uitgezet. Dit is het meest gebruikelijke in de wiskunde. Om dezelfde redenen noemen we de verticale as de Y-as. De X-as en de Y-as SAMEN noemen we het assenstelsel. Daar waar de assen elkaar snijden, is de oorsprong, meestal aangeduid met de hoofdletter O. In een tekening: zie tek.1 Het punt P heeft als coördinaten (x,y), wat zoiets betekent als 'x' naar rechts en 'y' naar boven. Wiskundiger P(x,y).
Om even terug te komen op ons voorbeeld:- de tijdas wordt de X-as, en de tijd wordt 'x'; - de snelheidsas wordt de Y-as, en de snelheid wordt 'y'.-Punt Q heeft als coördinaten (15,45): Q(15,45). In een grafiek is het altijd zo, dat er een bep. verband bestaat tussen de x- en de y-waarden. In ons voorbeeld: De snelheid is 'y' km/uur na 'x' sec. Dit verband noemen we in de wiskunde een relatie, dus y staat in reletie met x. De wiskundige notatie: yRx. Een andere relatie tussen y en x is de volgende: y=x+3 dus als x=1 dan y=4, als x=-3 dan y=0, enz. Dit verband is weer grafisch weer te geven, zie fig.2.

Een vraag als we een grafiek gaan tekenen is, welk gedeelte van de grafiek willen we zien? Zie fig.3. M.a.w. als we het gebied tussen de dikke strepen willen zien, moeten we x1 en x2 weten. Dit gebied tussen x1 en x2 noemen we het domein. Bij een gegeven domein horen de nodige y-waarden. In ons vb. is het domein van 0 tot 55 (sec). De y-waarden zijn dan min. 0 en max. 75 (km/uur). Dit gebied van 0 tot 75 noemen we het bereik van de grafiek. Zie tek.3.
Door wat te oefenen met bovenstaande begrippen zult u ze weldra kunnen gebruiken is het programma.
Voordat u het programma gaat 'runnen' moet ik eerst iets vertellen over een bijzondere relatie: de functie. Wat is een functie? Een functie is een relatie waarbij aan elke x-waarde hooguit één y-waarde wordt gekoppeld. Onderstaande grafieken zijn functies.



Want er is geen één x waarbij meer als één y hoort. Onderstaande figuren zijn geen functies.



Nu bent u zover dat u het programma kunt gebruiken. Het programma geeft informtie welk domein u het laatst hebt gebruikt, welk bereik u het laatst heeft gebruikt en of u ja dan nee een andere functie wilt invoeren. Verder wordt de hoogte van het scherm gelijk gesteld aan het opgegeven bereik en de breedte van het scherm wordt gelijk gesteld aan het opgegeven domein.
Enkele grafieken om mee te beginnen:

%Y=%X domein -10 tot 10
bereik -10 tot 10

%Y=%X.%X domein -10 tot 10
bereik 0 tot 100 (grafiek past net)
of bereik -10 tot 110 (mooier)

%Y=%X.2 domein -10 tot 10
bereik -10 tot 10

%Y=SIN(%X) domein -2.PI tot 2.PI
bereik -1 tot 1 (grafiek past net)
of bereik -1,2 tot 1,2 (mooier)

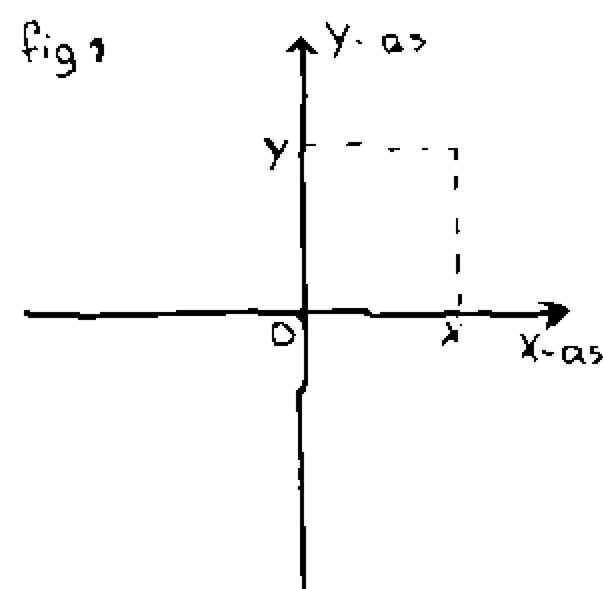
Je ziet, het bereik kun je beter iets ruimer opgeven, om zodoende een

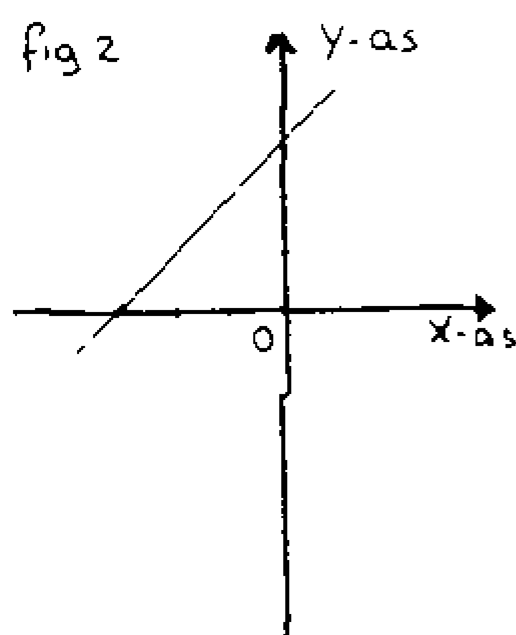
mooier plaatje te krijgen. Het programma kan zelf ook het bereik uitrekenen, als u dat zelf niet zo vlug kunt. Toets dan een bereik van 1 tot 1 in, en het programma zorgt dan dat de grafiek keurig in het beeld blijft en nergens de boven- of onderkant van het scherm raakt.
Achteraf (na RUN) zegt het programma ook nog wat voor bereik hij zelf had uitgerekend. Probeer bv. eens:

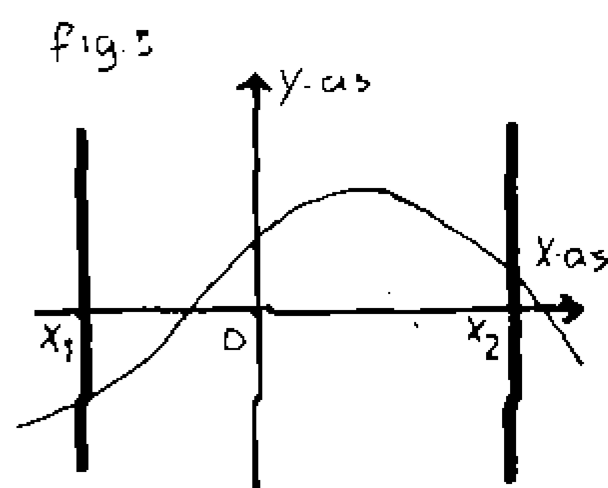
```
%Y=(%X+7).(%X+30.(%X-5).%X    domein van -6 tot 6  bereik van 1 tot 1
```

Antoin Reijnen.

Voor de eerste keer: type dan op vraag 270 JA en op vraag 290 (bv.)
600%Y=%X dan komt vraag 270 nogmaals, type dan NEE.







Grafieken 2. ******

Van A.van Wijngaarden kwam een reactie op grafieken van A.Reijnen. Hij vond het jammer dat je 2 keer moest runnen. Door hem werd het volgende veranderd of toegevoegd:

```
1 Z=CH"b";T=TOP;DO T=T-1;U.?T=13 AND T?3=Z      zoek funktie regel
2 Z=T+4;IN."functie"$Z                          $Z wordt functie regel
3F.Y=(Z+L.Z) TO (Z+58);?Y=32;N.                 de rest na de funktie
200b%Y=%X                        de rest spaties              spaties
210 R.
```

Hij zet dus string Z direkt achter regelnr.200b Z wordt dus een functie-string. Regel 10 zoals eerst laat hij dus weg.
De funktie %Y=1/%X geeft een ERROR, daarom het volgende

```
4 DIMB15;$B="G.w"                             foutstring
5 ?16=B;?17=B/256                             gezet
130 %X=%A;T=3                                 T geeft aan waarin fout
131eDO
132 GOS.b;PLOT 5,%(%X/%S+%E),%(%Y/%T+%F)
150a%X=%A;T=1                                 fout zit ergens anders
151gGOS.b;%Z=%Y;%Q=%T;T=0                     fout weer ergens anders
152fDO
153 GOS.b
191 G.d
280w%X=%X=%S                      zet %X eentje verder
290 IF T=0 G.f                    je ziet aan T waarheen terug sprin-
300 IF T=1 G.g                                                   gen
310 G.e
```

En het volgende: 80dCLEAR 4   en regel 70 niet afsluiten met GOS.a maar met G.a

Afstandsberekening. "AFSTANDS BER." ********
Dit programma berekent de afstand in kilometers tussen twee plaatsen op de aarde, welke worden aangeduid met de, door zeldamateurs, gebruikte QTH-locator. Toelichting op dit programma, van J.H.Schoon:
Door bij de vraag naar de QTH-locator van het tegenstation RETURN te geven, wordt het programma beëindigd.

MORSE encoder/decoder. ***
Door Kuipers uit Amstelveen werd een interface (schema +program) ingestuurd dat werkt, maar voor verbetering vatbaar is. knelpunten bij al dergelijke interfaces is de gevoeligheid van de 'fading' en voor storingen. Dit schema, bewerkt voor de Atom, is naar een oorspronkelijk ontwerp van Immerzeel (Radio Bulletin 7-1979). Het programma staat op een bandje in het bandjes-archief.

RTTY. ***
Behalve MORSE-signalen, kun je ook andere gegevens uit de lucht plukken en via de Acorn gedecodeerd over het scherm laten rollen. Zo ook met TELEX. Telex maakt echter geen gebruik van de Morse-code in 'lang''kort' 'pauze' (.- is bv. a) maar van de zgn. 'Murray-codering'. Alle karakters bestaan uit 5 evenlange elementen die elk 'hoog' of 'laag' kunnen zijn. Bij telex per lijn is dan 'hoog': er loopt een stroom, 'laag': geen stroom. Bij Radio-TTY verschillen hoog en laag in de toonhoogte van het signaal in het audio-gebied. Men gebruikt de frequenties 2125 Herz en 2295 Herz. Zo gecodeerd wordt een a hhlll De karakters worden voorafgegaan en gevolgd door start en stopteken, het zgn. 'start-bit' en 'mark'. Verder is er nog een 'cijfershift' anders kom je met 5 elementen (= 32 tekens max.) niet uit. Het program werd ingezonden door: Jaap v/d Woestijne. De listing staat op het zelfde bandje als voor de Morse, het schema is overgenomen uit Funkschau 3/1981.

Beschrijving FAX-decoder. *********
FAX is een afkorting van FACSIMILE, wat reproduktie van prenten betekent. In de telecommunicatie wordt facsimile aangeduid met code F4, het is dus een vorm van frequentie modulatie. Facsimile wordt door sommige kortegolf stations gebruikt om weerkaarten voor de scheepvaart uit te zenden. Op de schepen bevindt zich een speciale FAX-ontvanger. Het systeem berust net als TV op het lijn voor lijn aftasten van de tekening. De lichte en donkere delen zorgen voor de frequentie verschuiving. 400 Hz naar boven is wit, 400 Hz naar beneden is zwart. Omdat bij weerkaarten alleen sprake is van wit of zwart, kan het signaal digitaal verwerkt worden. Deze weerkaarten moeten niet verward worden met satelliet foto's. Ze bevatten gegevens over luchtdruk verdeling, ligging van fronten, hoogte van de golven van de zee enz.
Interface:
Om de frequentie verschuivingen om te zetten in 0 en 1 is een interface nodig. Hiervoor is een IC TBA120 toegepast, dit is een fasedecoder. Het is de bedoeling dat de kortegolf ontvanger d.m.v. de BFO een fluittoon produceert, deze toon zal hoger of lager worden als de zender in frequentie wordt gemoduleerd. Die variaties worden faseverschillen als een LC-kring in resonantie wordt gebracht. Deze faseverschillen worden door de TBA120 gedetecteerd. De output bevat ook de dubbele ingangs frequentie Die wordt verwijderd met een dubbel RC-filter. Dan volgt een opamp bv. 747 of LF356, die geschakeld is als comperator. Het signaal gaat dan als

Versterker Filter Gelijkrichter Sch.tr.

IC1,2,3 = LF356

Schema van de MORSE-decoder

Schema van de RTTY-decoder

Schema voor de FAX-decoder

0V of 5V naar de computer. Deze moet dan in het juiste tempo witte of zwarte punten op het scherm zetten. Bovendien moet op het juiste moment op een nieuwe lijn begonnen worden. Als het scherm vol is kan met het programma GDUMP van A.Toorman het plaatje op de printer afgedrukt worden. Toelichting bij het schema zelf:
L1 vormt samen met C1 een resonantie kring van ca. 2000 Hz, bv. Amroh F4. Bij een andere spoel eventueel C1 aanpassen. De ingang komt aan de bandrecorder uitgang van de ontvanger. De uitgang komt aan pootje 7 van de DIN connector van de Acorn Atom.
Problemen:
Het oplossend vermogen hangt af van het aantal beeldpunten per mm (hor) en het aantal lijnen per mm (vert). Het benodigde geheugen hangt af van het aantal beeldpunten per lijn en het aantal lijnen per tekening. Om te kunnen zien of alles goed gaat moet alles zich af spelen in het gebied #8000 tot #9800. Hierin passen 192 lijnen met 256 punten. FAX-zenders leveren meestal 120 lijnen per min. Het scherm is dus in 1,5 min. vol. Een uitzending duurt vaak wel 15 min. of 1800 lijnen. Dit doet vermoeden dat horizontaal toch ook minstens 500 tot 1000 punten nodig zullen zijn. Dat klopt dan ook want als we een lijn op een lijn afbeelden krijgen we een zijdelings sterk samengedrukt plaatje op het scherm. Door nu bij het printen elk punt 2x af te drukken wordt het plaatje in de breedte uitgerekt. Het grappige is nu dat ondanks dat we schandelijk hebben gesnoeid in detail weergave er toch zo nu en dan een goed herkenbaar plaatje ontstaat.
Lijst van FAX-zenders:
Groot Brittanie: Bracknell

roepletters	frequentie	uitzendtijden(GMT)
GFA21	3289.5	0000-2400
GFA22	4610	"
GFA23	8040	"
GFA24	11086.5	"
GFA25	14536	"
GFE25	2618.5	1900-0500
GFE21	4782	0000-2400
GFE22	9203	"
GFE23	14436	"
GFE24	18261	0500-1900

W.Duitsland: Hamburg/Quickborn/Pinneberg

roepletters	frequentie	uitzendtijden(GMT)
DGC70L5	3695.8	0905, 0918*, 0931*, 0948*, 1002, 1020, 1555, 1728, 2108, 2133, 2145, 2200, 2226, 2250 h *: niet op zon- en feestdagen
DGN62H6	13627.1	zie hier boven

Zelf (H.Kuiper uit Amstelveen) heb ik geexperimenteerd met 4782, 8040 en 9203 kHz. Hierop worden enkele kaarten per uur uitgezonden, zodat er altijd wel wat te ontvangen is.
Het bijbehorende programma staat weer op het zelfde bandje als MORSE en RTTY, zie bandjes-archief.